

Self-training for multi-target regression with tree ensembles



Jurica Levatić^{a,b,*}, Michelangelo Ceci^c, Dragi Kocev^{a,b,c}, Sašo Džeroski^{a,b}

^a Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia

^b Jožef Stefan International Postgraduate School, Ljubljana, Slovenia

^c Department of Computer Science, University of Bari Aldo Moro, Bari, Italy

ARTICLE INFO

Article history:

Received 26 July 2016

Revised 10 February 2017

Accepted 11 February 2017

Available online 12 February 2017

Keywords:

Semi-supervised learning

Self-training

Multi-target regression

Random forests

Reliability of predictions

Predictive clustering trees

ABSTRACT

Semi-supervised learning (SSL) aims to use unlabeled data as an additional source of information in order to improve upon the performance of supervised learning methods. The availability of labeled data is often limited due to the expensive and/or tedious annotation process, while unlabeled data could be easily available in large amounts. This is particularly true for predictive modelling problems with a structured output space. In this study, we address the task of SSL for multi-target regression (MTR), where the output space consists of multiple numerical values. We extend the self-training approach to perform SSL for MTR by using a random forest of predictive clustering trees. In self-training, a model iteratively uses its own most reliable predictions, hence a good measure for the reliability of predictions is essential. Given that reliability estimates for MTR predictions have not yet been studied, we propose four such estimates, based on mechanisms provided within ensemble learning. In addition to these four scores, we use two benchmark scores (oracle and random) to empirically determine the performance limits of self-training. We also propose an approach to automatically select a threshold for the identification of the most reliable predictions to be used in the next iteration. An empirical evaluation on a large collection of datasets for MTR shows that self-training with any of the proposed reliability scores is able to consistently improve over supervised random forests and multi-output support vector regression. This is also true when the reliability threshold is selected automatically.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

The major machine learning paradigms are supervised learning (e.g., classification, regression), where all the data are labeled, and unsupervised learning (e.g., clustering), where all the data are unlabeled. Semi-supervised learning (SSL) [1] examines how to exploit both labeled and unlabeled data, aiming to benefit from the information that unlabeled data bring. SSL is of a practical relevance because, in many real-world scenarios, labeled data are scarce due to a costly and/or time-consuming labelling procedure; while unlabeled data abound and are easy to obtain. For example, such scenarios are encountered in life sciences (gene function prediction, quantitative structure-activity relationship modelling), ecology (habitat and community modeling), multimedia (annotation of images and videos) and semantic web (categorization and analysis of text and web).

Intuitively, SSL yields best results when there are few labeled examples as compared to unlabeled ones (i.e., large-scale labelling

is not affordable). Such a scenario is especially relevant for machine learning tasks with structured outputs where, due to the increased complexity of the output, labelling of the data is even more difficult. Consider, for example, the problem of natural language parsing, where the aim is to predict the parse tree that generates a given input sentence. To label the data, linguists need to determine the parse tree for input sentences. This is feasible for a few sentences, but for large number of sentences the process becomes very tedious and expensive: It took 2 years to manually construct parse trees for 4000 sentences of Penn Chinese Treebank [2]. At the same time unlabeled input sentences are readily available in vast amounts. Another prominent example comes from the ecological modelling domain, where some attribute values are easily available (e.g. temperature, humidity) whereas some other attribute values have to be manually collected/measured by experts and thus, can be the subject of the prediction process (e.g. water pollution in a river, or abundance of specific species which populate the river). Obviously, in the latter case, data collection is very expensive and time consuming, so only few observations can be obtained with limited resources [3].

In this study, we are concerned with SSL for the task of *multi-target regression* (MTR). MTR is a structured output prediction task

* Corresponding author.

E-mail addresses: Jurica.Levatic@ijs.si (J. Levatić), michelangelo.ceci@di.uniba.it (M. Ceci), Dragi.Kocev@ijs.si (D. Kocev), Saso.Dzeroski@ijs.si (S. Džeroski).

where the goal is to predict multiple continuous target variables (also known as multi-output or multivariate regression). In many real-life problems, we are interested in simultaneously predicting multiple continuous variables. Prominent examples of this task come from ecology: predicting the abundance of different species occupying the same habitat [4], assessing different properties of forests [5], or estimating vegetation quality indices [6]. We argue that SSL, as for classical machine learning tasks, can lead to improved predictive capabilities also for MTR by leveraging the contribution of unlabeled examples and, at the same time, by exploiting the possible dependencies among the multiple target variables.

The handful of existing SSL methods for structured output prediction almost exclusively deal with discrete outputs. Here, a prominent work was done by Brefeld [7], who used the co-training paradigm and the principle of maximizing the consensus among multiple independent hypotheses to develop semi-supervised support vector learning algorithm for joint input-output spaces and arbitrary loss. Zhang and Yeung [8] proposed a semi-supervised method based on Gaussian processes for a task related to MTR: multi-task regression. In multi-task learning the aim is to predict multiple single-target variables with different training sets (in general, with different descriptive attributes) at the same time. The few existing SSL methods for MTR are highly specialized for individual applications. For example, Navaratnam et al. [9] have proposed a SSL method for MTR specialized for computer vision. On the other hand, SSL for single-target regression has received more attention in the past [10–13]. While it is possible to decompose a MTR problem into several (local) single-target ones and use such methods, there are several advantages of learning a global multi-target model over learning a separate local model for each target variable. Global models have better computational efficiency, typically perform better and overfit less than a collection of single-target models [14,15].

We propose a global SSL method for MTR. More specifically, we extend the self-training approach [16] to the task of MTR. The main advantage of this iterative SSL approach is that it can be “wrapped” around any existing (supervised) method. In the past, several studies have proposed supervised methods for solving the task of MTR directly and demonstrated their effectiveness [6,14,17,18]. We propose to use predictive clustering trees (PCTs), or more precisely, random forests [19] of PCTs for MTR, as base predictive models [14] for the self-training approach. PCTs are a generalization of standard decision trees towards predicting several types of structured outputs: tuples of continuous/discrete variables, hierarchies of classes, and time series.

The main principle of self-training is iterative usage of its own most reliable predictions for the unlabeled data as additional data in the training process. The most reliable predictions are selected by applying a threshold on the reliability scores of predictions. A good reliability scoring function assigns a high score to the predictions with low error and a low score to the predictions with high error. Obviously, a proper reliability scoring function is crucial for the success of self-training, since an error once made can reinforce itself in the subsequent iterations. However, developing a good reliability scoring function is not a simple task [20]: This has not yet been entirely resolved in the single-target regression and classification, and even less for the task of MTR.

In this paper, we propose and evaluate several reliability scoring functions for MTR, which are based on the mechanisms provided by ensemble learning. Namely, we use the variance of the votes of an ensemble and random forest proximities to estimate the reliability of predictions [19,20]. These reliability estimates are by-products of ensemble learning. Hence, they impose almost no additional computational overhead, as opposed to some other reliability estimates for regression [20]. This aspect is especially important in SSL, where we can expect to deal with huge amounts

of unlabeled data and/or to re-train the model several times (as in self-training). In order to empirically determine the performance limitations of the proposed approach to self-training for MTR, we use oracle scoring (the best possible scoring function) and random scoring as benchmark scoring functions. Finally, we explore the influence of two strategies for merging per-target scores into a global score (normalized averaging and ranked averaging) on the performance of self-training.

We also consider the problem of automatically determining the threshold on the reliability scores of predictions. The thresholding is crucial for the selection of the unlabeled examples with the most reliable predictions. The selected unlabeled examples together with the predictions are then considered as training examples in the next iteration. To this end, we propose an automatic threshold selection algorithm for SSL that exploits the out-of-bag error obtained when learning the ensemble.

Our study investigates three important questions: (1) Can unlabeled data improve predictive performance on MTR tasks in a self-training setting? (2) Which reliability scoring function yields the best predictive performance in this setting? (3) Can we exploit the advantage introduced by the self-training setting for MTR when an automatic threshold selection algorithm is used? To address these questions, we perform experimental evaluation of self-training with the various reliability scoring functions using 9 MTR datasets from various domains. The evaluation reveals that self-training, coupled with any of the proposed reliability scoring functions, is able to outperform a supervised random forest and multi-output support vector regression (MSVR) [21,22]. In particular, all of the proposed reliability scoring functions performed better than random scoring. The best results, excluding the upper (oracle) limits on the performance of self-training, were achieved by using a reliability score based on the variance of the votes of ensemble members.

We summarize the major contributions of this paper as follows:

- A SSL method tailored for the task of MTR based on the predictive clustering framework for predicting structured outputs.
- Two reliability scoring functions for MTR predictions, two normalization strategies, and two strategies for merging per-target scores into a global score.
- Empirical determination of the upper bounds on the performance, i.e., the potential of SSL with self-training.
- An automatic threshold selection algorithm, i.e., a practical solution for exploiting the potential of SSL with self-training.
- Empirical evaluation of the proposed method on 9 MTR datasets.

An initial investigation of the proposed SSL method for MTR has been presented in a workshop paper [23,24]. We extend that work along six major dimensions. First, we propose a new reliability scoring function based on the random forest proximities, in addition to the one based on the variance of the votes of the ensemble members. Second, we propose four strategies for obtaining a single global score from the per-target scores. Third, we propose an oracle score to test the performance bounds of the self-learning paradigm. Fourth, we propose an algorithm for the automatic selection of the threshold for reliability of predictions. Fifth, we consider a more sophisticated stopping criterion for self-training, which automatically stops learning if the performance begins to degrade. Finally, the empirical evaluation is performed on a larger collection of MTR datasets.

The remainder of the paper is organized as follows. In the next section, we present the background of the work presented here. This includes a discussion on related work on the topics of MTR and SSL, and a brief description of the predictive clustering framework. Next, in Section 3 we describe the self-training approach and the proposed reliability scores for MTR. The experimental design

and key experimental questions are outlined in Section 4. The results of the empirical investigations are presented and discussed in Section 5. Finally, we draw the main conclusions and give directions for further work in Section 6.

2. Background

2.1. Related work

Within SSL, there are different classes of algorithms [16]: low-density separation methods, graph-based methods, generative models, and methods based on iterative training. The last group of methods consists of two main paradigms: self-training and co-training. These paradigms have established themselves as the main approaches to SSL for practical reasons: They are based on a convenient extension of existing supervised methods towards SSL.

Self-training [16] and co-training [25] are iterative approaches that, in addition to labeled data, use unlabeled data and their own predictions of its labels, in the learning process. The latter approach trains models on independent feature sets (i.e., views): These models help one another to avoid biasing to their own prediction errors. In this paper, we are concerned with the self-training paradigm and thus introduce it in more detail.

Self-training was first proposed by Yarowsky [26] for word sense disambiguation. Since then, it has had a number of successful applications, including detection of objects in images [27,28], identification of subjective nouns [29] and learning human motion over time [30]. Abney [31] and Haffari & Sarkar [32] proposed a theoretical analysis of the self-training algorithm. It showed that the self-training framework minimizes the same objective function of the base model (but in a different way). In the specific case considered by the authors it minimizes a cross-entropy based objective function, given that the base model reduces the same objective function.

An overwhelming majority of studies relying on self-training to perform SSL are concerned with the task of classification. However, there are handful of methods based on self-training (or co-training) for the task of (single-target) regression [10–13]. Recently, Sousa and Gama [33] proposed a self-training method based on adaptive model rules for MTR from data streams [34]. To the best of our knowledge, self-training has not yet been implemented for the task of MTR in the batch learning setting.

Multi-label classification is a machine learning task somewhat related to MTR. There the goal is to simultaneously predict multiple binary labels, while in MTR the goal is to simultaneously predict multiple continuous variables. Contrary to MTR, several semi-supervised methods for multi-label classification were previously published [35–37]. However, such methods are not directly applicable for MTR, and are thus not comparable to the method proposed in this paper.

Independently of the self-training paradigm, MTR has recently received increasing attention [38]. An early method for MTR was proposed by Brown and Zidek [39] who adapted the standard ridge regression to multivariate ridge regression. Later, Breiman and Friedman [40] proposed the Curds&Whey method, where correlations between the target variables are modelled in a post-processing phase. Recently, several machine learning methods, popular for regression, have been implemented also for the task of MTR, such as decision trees [17,18], support vector machines [21,22,41] and k-nearest neighbours [42]. Ensemble methods for MTR include rule ensembles [43], bagging and random forests [14]. A different approach is followed by Tsoumakas et al. [44], who proposed another ensemble-based method for MTR, where new target variables are constructed as random linear combinations of existing target variables.

In principle, any of the listed MTR methods could be used as the base model of self-training for MTR – given that a measure for the reliability of its predictions is defined. As already noted in the introduction, our ideas for development of reliability scores are based on the mechanisms provided by ensemble learning, and in particular by random forests. This is the main reason why we choose random forests of PCTs for MTR [14] as base models for self-training. As self-training relies on its own predictions, a method with state-of-the-art predictive performance, such as random forests, is needed. Moreover, random forests of PCTs were found to perform better than or comparable to other ensemble approaches for MTR, while being computationally efficient [43,44].

In this study, we propose several scoring functions for the reliability of MTR predictions based on the studies of Bosnić and Kononenko [20] and Briesemeister et al. [45]. The first extensively studies different reliability scores for regression, based on sensitivity analysis, local cross-validation, analysis of the density of the distribution of learning examples, and the variance of bagged models. The empirical evaluation highlighted the variance of bagged models as the most successful reliability score. The second study proposes a reliability score for regression predictions based on estimating the examples' error by considering its local environment in the training set. In a similar fashion, our reliability score also considers the neighbourhood of the examples as defined by the distance function intrinsic to random forests, and estimates errors with out-of-bag examples.

2.2. Ensembles of predictive clustering trees for MTR

The basis of the semi-supervised method proposed in this study is the use, in an ensemble learning fashion, of PCTs. In this section, we first briefly describe PCTs for MTR, and then the method for learning a random forest of PCTs. Both are described in more detail in Kocev et al. [14].

2.2.1. Predictive clustering trees for MTR

The PCT framework views a decision tree as a hierarchy of clusters, where the top-node corresponds to one cluster containing all data. This cluster is recursively partitioned into smaller clusters while moving down the tree. The PCT framework, including PCTs for MTR and ensembles thereof, is implemented in the CLUS system [14], available at <http://sourceforge.net/projects/clus>.

PCTs are induced with a standard *top-down induction of decision trees* (TDIDT) algorithm, which takes as input a set of examples E and outputs a tree. The heuristic that is used for selecting the tests to put in internal tree nodes is the reduction in variance caused by partitioning the examples according to the tests. By maximizing the variance reduction, the cluster homogeneity is maximized and the predictive performance is improved.

The main difference between the algorithm for learning PCTs and a standard decision tree learner is that the former considers the variance function and the prototype function (that computes a label for each leaf) as *parameters* that can be instantiated for a given learning task. So far, PCTs have been instantiated for the following tasks [14]: multi-target prediction (which includes MTR), hierarchical multi-label classification and prediction of time-series. In this article, we focus on the task of MTR.

The variance and prototype functions of PCTs for MTR are instantiated as follows. The variance is calculated as the sum of the variances of the target variables, i.e., $Var(E) = \sum_{i=1}^T Var(Y_i)$, where T is the number of target variables, and $Var(Y_i)$ is the variance of target variable Y_i . The variances of the targets are normalized, so each target contributes equally to the overall variance. The normalization is performed by dividing the estimates with the standard deviation for each target variable on the available training set. The prototype function (calculated at each leaf) returns as a prediction

the vector of mean values of the target variables, calculated by using the training examples that belong to the given leaf.

2.2.2. Ensembles of predictive clustering trees

We consider random forests of PCTs for structured output prediction, as implemented by Kocev et al. [14] in the CLUS system. The individual PCTs in a random forest are constructed by the approach proposed by Breiman [19]. A random forest is an ensemble of trees, where diversity among the predictors is obtained by using bootstrap replicates of the training set (as in bagging) and by changing the set of descriptive attributes during learning. Bootstrap samples are obtained by randomly sampling training examples, with replacement, from the original training set, until an equal number of examples as in the training set is sampled. Breiman [46] showed that bagging can give notable gains in predictive performance, when applied to unstable learners (for which small changes in the training set result in large changes in the predictions), such as classification and regression tree learners.

To learn a random forest, the classical PCT algorithm for tree construction is randomized by replacing the standard selection of attributes with a randomized selection. More precisely, at each node in a decision tree, a random subset of the descriptive attributes is taken, and the best attribute is selected from this subset. The number of attributes that are retained is given by a function f of the total number of descriptive attributes D (e.g., $f(D) = \lfloor \log_2(D) + 1 \rfloor$).

In random forest of PCTs, the prediction for a new example is obtained by combining the predictions of all PCTs in the forest. For the MTR task, the prediction for each target variable is computed as the average of the predictions for that target variable obtained from each tree in the forest.

3. Self-training for MTR with ensembles of PCTs

In this section, we present our approach to semi-supervised learning for the task of MTR. The approach is based on the predictive clustering framework and the self-training paradigm. In a nutshell, the proposed approach works as follows. To begin with, a predictive model (i.e., a random forest of PCTs) is constructed by using all of the available labeled examples (the training set). Next, the predictive model is applied to the unlabeled examples, i.e., it produces labels for the unlabeled examples. The examples with the most reliable predictions are then selected and added to the training set. Next, a new predictive model is constructed using the updated training set. The learning algorithm continues until a stopping criterion is satisfied. The pseudo-code of the self-training algorithm for MTR with ensembles of PCTs (named CLUS-SSL) is outlined in Table 1.

In the description of the CLUS-SSL algorithm, three key notions require a precise definition: the reliability scoring function, the threshold on the reliability scores and the stopping criterion. First, self-training relies strongly on the assumption that its own (most reliable) predictions are correct. Hence, the most crucial part of the algorithm is to define an appropriate reliability scoring function. A good reliability scoring function should be able to discern correct predictions (thus assigning them high scores) from wrong predictions (thus assigning them low scores).

We define the reliability score of a predictive model \mathcal{M} as a function:

$$\text{Reliability}^{\mathcal{M}} : E_u \rightarrow [0, 1], \quad (1)$$

where E_u is a set of unlabeled examples. For a given example $e \in E_u$, $\text{Reliability}^{\mathcal{M}}(e)$ denotes an estimate of the probability that the prediction $\mathcal{M}(e)$ is correct, i.e., the reliability score is a kind of a proxy for the accuracy of the prediction.

Table 1

The learning algorithm for self-training with random forests of PCTs (CLUS-SSL). Here, E_l is a set of the labeled examples, E_u is a set of unlabeled examples, k is the number of trees in the forest, D is the total number of descriptive attributes, $f(D)$ is a function which gives the size of the feature subset for random forests considered at each node during tree construction, and τ is the threshold for the reliability of predictions.

```

procedure CLUS-SSL( $E_l, E_u, \tau, k, f(D)$ )
returns Forest
1: repeat
2:    $F = \text{RForest}(E_l, k, f(D))$ 
3:    $E'_u = \text{predict}(F, E_u)$ 
4:   for each  $e_u \in E'_u$  do
5:     if  $\text{Reliability}^F(e_u) \geq \tau$  then
6:       move  $e_u$  from  $E'_u$  to  $E_l$ 
7:       drop  $e_u$  from  $E_u$ 
8: until stopping_criterion()
9: return  $F$ 

```

Next, a user-defined threshold ($\tau \in [0, 1]$) on the reliability score needs to be set to select the unlabeled examples that should be added to the training set. If the reliability of the prediction for an unlabeled example is greater than τ , the example is moved from the unlabeled set (E_u) to the training set (E_l), together with the predicted values of its target variables. This procedure is iterated until the stopping criterion is met (see Section 3.5 for more details on stopping criteria).

Within each iteration of the self-training algorithm (see Table 1), we solve a MTR problem with m continuous targets. The existing reliability scoring functions are tailored for single-target problems [20]. We propose to calculate the reliability scores for a given unlabeled example as follows. We first calculate m per-target reliability scores, i.e., we apply a reliability scoring function to each of the targets separately. Next, the m reliability scores are aggregated into a single reliability score. The threshold τ is then applied to the aggregated reliability score. We explore two aggregation schemes for the per-target reliability scores: averaging-based and minimum-based aggregation. The averaging-based scheme consists of simple averaging of the per-target reliability scores, whereas the minimum-based aggregation scheme is more conservative. There, the examples' prediction is considered as reliable as the prediction of its least reliable target. The two aggregation schemes are defined as follows:

$$\text{Reliability}_{\text{Avg}}(e) = \frac{1}{m} \sum_{i=1}^m \overline{\text{Reliability}}_i(e), \quad (2)$$

$$\text{Reliability}_{\text{Min}}(e) = \min_{i=1, \dots, m} (\overline{\text{Reliability}}_i(e)), \quad (3)$$

where $\overline{\text{Reliability}}_i(e)$ is a normalized (in $[0, 1]$) reliability score for the i th target variable of an example e .

We consider two strategies of normalization of per-target scores: min-max normalization and ranking-based normalization. Formally:

$$\begin{aligned} \overline{\text{Reliability}}_i^{\text{Norm}}(e_u) &= \frac{\text{Reliability}_i(e_u) - \min_{j=1, \dots, |E_u|} \text{Reliability}_i(e_j)}{\max_{j=1, \dots, |E_u|} \text{Reliability}_i(e_j) - \min_{j=1, \dots, |E_u|} \text{Reliability}_i(e_j)}, \end{aligned} \quad (4)$$

$$\overline{\text{Reliability}}_i^{\text{Rank}}(e_u) = 1 - \frac{\text{Rank}_i(\text{Reliability}_i(e_u)) - 1}{|E_u| - 1}, \quad (5)$$

where $\text{Reliability}_i(e_u)$ is the reliability score for the i th target variable of the example e_u , $|E_u|$ is the number of unlabeled examples, and the Rank_i function gives ranks to unlabeled examples according to the reliability scores of the i th target variable.

Table 2

An illustrative example of using the procedure for calculating of the reliability scores according to different aggregation and normalization schemes. The same examples e_1 , e_2 and e_3 are used in both the upper and lower part of the Table.

| Normalized per-target reliability scores | | | | |
|--|-----------------------------|-----------------------------|--------------------------------|--------------------------------|
| Reliability scores | | | Aggregated scores | |
| $Reliability_1^{Norm}$ | $Reliability_2^{Norm}$ | | $Reliability_{1,2}^{Norm Avg}$ | $Reliability_{1,2}^{Norm Min}$ |
| e_1 | 0.8 | 0.1 | $\frac{0.8+0.1}{2} = 0.45$ | 0.1 |
| e_2 | 0.3 | 0.5 | $\frac{0.3+0.5}{2} = 0.4$ | 0.3 |
| e_3 | 0.9 | 0.4 | $\frac{0.9+0.4}{2} = 0.65$ | 0.4 |
| Ranked per-target reliability scores | | | | |
| Reliability scores | | | Aggregated scores | |
| $Reliability_1^{Rank}$ | $Reliability_2^{Rank}$ | | $Reliability_{1,2}^{Rank Avg}$ | $Reliability_{1,2}^{Rank Min}$ |
| e_1 | $1 - \frac{2-1}{3-1} = 0.5$ | $1 - \frac{3-1}{3-1} = 0$ | $\frac{0.5+0}{2} = 0.25$ | 0 |
| e_2 | $1 - \frac{3-0}{3-1} = 0$ | $1 - \frac{1-1}{3-1} = 1$ | $\frac{0+1}{2} = 0.5$ | 0 |
| e_3 | $1 - \frac{3-1}{3-1} = 1$ | $1 - \frac{3-0}{3-1} = 0.5$ | $\frac{1+0.5}{2} = 0.75$ | 0.5 |

The use of the ranking-based normalization strategy is motivated by the following. It can happen that the distributions of the per-target scores can be very different, thus introducing different biases to the aggregated reliability score. Ranking-based normalization, instead of considering the absolute values of the per-target reliability scores (as in min-max normalization), considers the relative differences between the per-target ranks. Issues related to the different distributions for different target variables are illustrated in Fig. 4 and discussed in more detail in Section 5.

In Table 2, we give an illustrative example of the procedure for calculating the reliability scores according to the previously defined aggregation schemes and normalization strategies. The task considered in this example is a MTR task with two target variables. Let us take three examples e_1 , e_2 and e_3 , with their per-target reliability scores (0.8, 0.1), (0.3, 0.5) and (0.9, 0.4), respectively. These scores are used to calculate four different reliability scores, with respect to the two different normalization schemes (min-max normalization and ranking-based normalization), and the two different aggregations schemes (averaging-based and minimum-based aggregation).

First, based on the (min-max) normalized reliability scores, we show the aggregation of the reliability scores using averaging-based and minimum-based aggregation. We see that with the averaging aggregation the examples have the order e_3 , e_1 , e_2 , while with the minimum aggregation the examples have the e_3 , e_2 , e_1 (as illustrated in the last two columns of the upper part of Table 2).

Second, using the ranking of the per-target reliability scores and Eq. (5), we produce the ranking-based per-target scores (illustrated in the first column in the lower part of Table 2). For the first target, the order of the examples is e_3 , e_1 , e_2 , while for the second target the ordering is e_2 , e_3 , e_1 . We next aggregate these scores using the two aggregation schemes and obtain the following orderings: for the averaging-based aggregation, the order is e_3 , e_2 , e_1 , for the minimum based aggregation e_3 is top-ranked, while e_2 and e_1 are tied.

The reliability scores for MTR ($Reliability_i(e)$) are obtained by exploiting some mechanisms provided directly by the ensemble learning paradigm. More specifically, we define two scores based on (1) the variance of the votes of the base predictive models in an ensemble and (2) random forest proximities [19]. We also propose two benchmark scores: a random and an oracle score. While the random score assigns random reliability scores to the examples, the oracle score assigns reliability scores based on the true labels of the unlabeled data (see Section 3.3 for more details).

These two benchmark scores determine the potential of the self-training approach and of the reliability scoring function. The

random score provides information on whether using reliability scores improves the predictive performance of self-training or not. If the use of a random reliability scoring function yields similar results as the use of a non-random reliability scoring function, this means that the reliability scoring function is not useful. The oracle score provides insight into whether the performance of self-training improves if the ‘true’ reliabilities of the predictions are given.

The four reliability scores (variance score, random forest proximity score, random score and oracle score), coupled together with the two aggregation schemes (avg and min) and two normalization schemes (norm and rank) yield to 13 reliability scores in total (the different aggregation/normalization schemes lead to the same results for the random score). The various reliability scoring functions are explained in more detail in the remainder of this section.

Finally, we emphasize that self-training is, in principle, a generic approach. We argue that the instantiation of self-training proposed in this study is effective specifically for the task of MTR, due to the (1) base model we use (i.e., random forest of PCTs), which was shown to successfully handle MTR [14], and (2) the proposed reliability scoring functions, developed specifically for the predictions of this base model. Obviously, self-training could be extended to types of structured outputs other than MTR, given appropriate base models are used, and reliability scoring function are properly defined for the prediction of these models.

3.1. Variance score

The variance score exploits the voting mechanism of the ensemble model [47]. Namely, when a prediction is made for an unlabeled example by a random forest, we consider it reliable if the predictions of the individual trees in the ensemble are coherent. This variance measure has been previously used in the context of bagging, where it was found to perform the best among a variety of approaches for estimating the reliability of regression predictions (in an extensive empirical comparison [20]).

In each iteration of our self-training approach, a random forest \mathcal{M} with k PCTs is built. The PCTs are trained on a set of labeled examples E_l and applied to a set of unlabeled examples E_u . First, for each unlabeled example $e_u \in E_u$, the per-target standard deviation r_u^i of votes of the ensemble is calculated as:

$$r_u^i = \sqrt{\frac{1}{k-1} \sum_{j=1}^k (\text{tree}_j^i(e_u) - \mathcal{M}^i(e_u))^2}, \quad i = 1, \dots, m, \quad (6)$$

where $\text{tree}_j^i(e_u)$ is the vote (i.e., prediction) for e_u returned by the j th tree for the i th target, \mathcal{M}^i is the prediction for e_u returned by the ensemble for the i th target (i.e., the average of the votes across all trees for the i th target), and m is the number of target variables.

In this case, in order to give high reliability scores for small standard deviation we define:

$$Reliability_i(e_u) = -r_u^i, \quad i = 1, \dots, m, \quad (7)$$

This reliability score can be theoretically motivated by the properties of variance. In fact, the variance represents (informally) how much the predictions differ from the mean. If we assume that the errors of the ensemble are distributed according to a normal distribution (this assumption is reasonable for most of the ensemble learning approaches and follows the central limit theorem [48]), r_u^i can be considered as a good approximation of the variance of the errors, which we can directly associate to the concept of reliability. Actually, our approach follows some previous studies [20] where the variance of predictions obtained by bagging of artificial neural networks was used to estimate reliability.

Combining this reliability score with the two normalization schemes and with the two aggregation schemes leads to four configurations, named *VarianceNormAvg*, *VarianceNormMin*, *VarianceRankAvg* and *VarianceRankMin*.

3.2. Random forest proximities score

This reliability scoring function relies heavily on the random forest inner mechanisms. More specifically, we exploit two such mechanisms: the measure of proximity between examples and the out-of-bag error estimation. A random forest has an intrinsic proximity measure [19]: For two examples e and f , proximity $p(e, f)$ is defined as the proportion of trees in the forest where e and f end in the same leaf.

When constructing ensembles from bootstrap replicates of the training data, the performance can be estimated using the out-of-bag error, i.e., the error on the examples that were not used to construct the individual base models. Breiman [49] presented empirical evidence that the out-of-bag error is a very accurate estimate of a nodes' errors in a regression tree. More generally, out-of-bag error is a good estimate of the error which an ensemble model will make on unseen examples.

We propose to join these two mechanisms (proximity between examples and out-of-bag error estimation) into a reliability scoring function. Namely, we extrapolate the expected error of unlabeled examples by using the out-of-bag errors of the labeled examples in their proximity. In this way, we obtain an estimate of the ensembles' predictive performance (and conversely error) for that example. In particular, we first compute the out-of-bag error for each labeled example and then, for each unlabeled example, we calculate its proximities to all the labeled examples. If the unlabeled example e_u often falls in the same leaf with a labeled example e_l (i.e., $p(e_u, e_l)$ is high), according to the above discussion, the out-of-bag error of the labeled example e_l is a good estimate of the error of the unlabeled example e_u . Thus, if the out-of-bag errors of labeled examples in the close proximity of the unlabeled example e_u are low, then it is expected that the error of e_u will also be low (i.e., the prediction is correct).

More formally, the per-target expected error ($expErr_u^i$) for each example ($e_u \in E_u$) is calculated as follows:

$$expErr_u^i = \sum_{e_l \in E_l} (p(e_u, e_l) \cdot OOBError_i(e_l)), \quad i = 1, \dots, m, \quad (8)$$

where $OOBError_i(e_l)$ is the out-of-bag error for the i th target of the labeled example e_l , m is the number of targets and $p(\cdot, \cdot)$ is the proximity function.

Our reliability scoring function based on the random forest proximities is similar to reliability estimation based on local neighbourhoods [20] – both functions consider the local neighbourhood of an example to estimate the reliability of its prediction. However, none of the scores there takes prediction errors into account. Briesemeister et al. [45] proposed a reliability estimate based on the errors in the local environment in the training set. This method is model-independent and requires estimation of errors on the training set, optimization of the number of neighbours and calculation of distances among examples. In contrast, the score proposed in this paper is tailored to random forests and adds no additional computational overhead, except for counting the number of times a pair of examples ended in the same leaf.

As an error measure for $OOBError_i(e_l)$, we use the root-mean-square error (RMSE). Moreover, in order to give high reliability scores for small out-of-bag errors, we define:

$$Reliability_i(e_u) = -expErr_u^i, \quad i = 1, \dots, m, \quad (9)$$

Combining this reliability score with the two normalization schemes and the two aggregation schemes leads to four configura-

tions, named as *RForestProxNormAvg*, *RForestProxNormMin*, *RForestProxRankAvg* and *RForestProxRankMin*.

3.3. Benchmark reliability scores

To assess the potential of the self-training paradigm and the reliability scoring function, we propose two benchmark scores: the random and the oracle scores. When self-training is performed with the random score, unlabeled examples are added to the training set in a random order. With the oracle score, they are added in the best possible order, that is, examples where the model makes the smallest actual error are added first. A good reliability score for self-training yields performance better than that of the random score, and as close as possible to the one of the oracle score. The *random score* assigns a random number between 0 and 1 to each unlabeled example.

For this purpose, we generate pseudo-random numbers according to the uniform distribution in $[0,1]$. New random scores are generated at each iteration of self-training and are assigned to the examples remaining in the unlabeled set.

The *oracle score* is defined on the basis of the actual error a predictive model makes on unlabeled examples. The SSL algorithms are typically evaluated by using the set of entirely labeled data, where unlabeled data are simulated by temporarily ignoring the labels. Knowing the exact labels of all data enables us to measure the errors made on the unlabeled data. Note that this score can not be used in practical applications of SSL. We use this score to assess how far the proposed reliability scoring functions are from the theoretically optimal score.

The oracle score is calculated as follows. In each iteration of self-training, predictions are made for the unlabeled examples. For each unlabeled example, we measure its per-target errors using the predicted and the real labels. The obtained per-target errors are then normalized to $[0, 1]$ the same way as in the other two reliability scores, so that small (per-target) error leads to high (per-target) reliability.

Random reliability scores and oracle reliability scores with the two normalization schemes and the two aggregation schemes lead to five configurations, named *Random*, *OracleNormAvg*, *OracleNormMin*, *OracleRankAvg* and *OracleRankMin*.

3.4. Automatic threshold selection

Once the predictions on unlabeled examples are made and reliability scores are calculated, it is necessary to choose the examples which will be considered as training examples in the next iteration (i.e., the ones with reliable predictions). For this purpose, a threshold on the reliability scores is needed. In Section 5, we demonstrate that a properly selected threshold is of great importance for the optimal performance of the self-training approach. If an inappropriate threshold is selected, such that it allows examples with erroneous predictions to enter the training set, the performance of self-training can deteriorate.

In this work, we propose an approach to automatically identify an appropriate threshold for the reliability of predictions. A simple solution, already proposed and empirically evaluated for simple binary classification in [50], is to select 10% of the most-reliable predictions and use the mean of their reliability score as a threshold. Henceforth, we will call this solution *Tavg10%*. A variant of this solution is to include the top 10% most-reliable predictions in the next iteration. Henceforth, we will call this solution *Ttop10%*. Note that, we adapted these procedures for the task of MTR.

A more sophisticated alternative, which we propose in this paper, is to simultaneously exploit the errors and reliability scores of out-of-bag examples. In particular, at each iteration of self-training, out-of-bag reliability scores for labeled examples are calculated.

Table 3

The learning algorithm for self-training with automatic threshold selection based on out-of-bag errors (CLUS-SSL-AUTO). Here, E_l is a set of labeled examples, E_u is a set of unlabeled examples, k is the number of trees in the forest, D is the total number of descriptive attributes, $f(D)$ is a function which gives the size of the feature subset for random forests considered at each node during tree construction, and *initial* is an indicator variable specifying whether the threshold is automatically selected only after the first iteration of throughout all the iterations.

| | |
|--|---|
| procedure CLUS-SSL-Auto($E_l, E_u, k, f(D), initial$) returns Forest 1: $E_l^{original} = E_l$ 2: repeat 3: $F = \text{RForest}(E_l, k, f(D))$ 4: $E'_u = \text{predict}(F, E_u)$ 5: if <i>initial</i> = <i>false</i> or (<i>initial</i> = <i>true</i> and isFirstIteration()) then 6: $\tau = \text{automaticOOB}(F, E_l^{original})$ 7: for each $e_u \in E'_u$ do 8: if $\text{Reliability}^F(e_u) \geq \tau$ then 9: move e_u from E'_u to E_l 10: drop e_u from E_u 11: until stopping_criterion() 12: return F | procedure automaticOOB(Forest, E) returns Threshold 1: $\text{OOBErrAll} = \text{getOOBErrors}^F(E)$ 2: $\text{reliabilityScores} = \text{Reliability}^F_{\text{OOB}}(E)$ 3: $\text{sortDescending}(\text{reliabilityScores})$ 4: for each $\tau' \in \text{reliabilityScores}$ do 5: $E' = \{e \in E \mid \text{Reliability}^F_{\text{OOB}}(e) \geq \tau'\}$ 6: $\text{OOBErrSubset} = \text{getOOBErrors}^F(E')$ 7: $\text{p-value} = \text{t-test}(\text{OOBErrAll}, \text{OOBErrSubset})$ 8: if $\text{p-value} < 0.05$ then 9: return τ' 10: return 0 |
|--|---|

Out-of-bag reliability scores of labeled examples are calculated in a similar way as the reliability scores of unlabeled examples, with the difference that, for each labeled example, only the trees for which that example was out-of-bag are considered. Then, for each candidate threshold on the reliability score, our approach evaluates if the mean of the out-of-bag error between the examples with reliability score greater than the considered threshold is *significantly different* (according to a statistical test) from the mean of the out-of-bag error of all the examples. Note that, in this procedure, we use only 'original' labeled examples, i.e., unlabeled examples which are added to the training set are not considered.

The basic idea is that, at each iteration, the algorithm should select the threshold such that examples with reliability scores greater than the threshold contribute to *significant* reduction in error. Candidate thresholds are all the out-of-bag reliability scores of labeled examples, taken in descending order. When the test accepts the hypothesis that the means are different, the algorithm stops and returns the last considered threshold. The threshold found in such a way is then applied to select unlabeled examples to be added to the training set. If statistical significance cannot be reached for any of the candidate thresholds, the algorithm returns 0, meaning that all of the unlabeled examples will be added to the training set at the next iteration.

The statistical test we consider is the two sample *t*-test for equal means (with a significance level of 0.05). In this algorithm for automatic threshold selection, we do not only consider trees (of the random forest) obtained at the last iteration, but all the trees obtained over all the considered iterations, thus guaranteeing stability of the threshold values. Henceforth, we will call this solution T_{OOB} and we will compare this solution with the variant $T_{\text{OOBInitial}}$ that determines the threshold only after the first iteration of the self-training framework, and uses this threshold throughout all the following iterations. The pseudo-code of the self-training method with the proposed algorithms for automatic threshold selection is presented in Table 3.

3.5. Stopping criteria

In this work, we consider two different stopping criteria for self-training. The first one is the commonly used stopping criterion which stops self-training if no unlabeled example is moved from the set of unlabeled examples (E_u) to the training set (E_l). This stopping criterion is satisfied when one of the following conditions is met: (1) the reliability scores for all the unlabeled ex-

amples are lower than the threshold, or (2) all unlabeled examples have been moved to the training set.

The second stopping criterion we consider, named *Airbag* [28], is designed to automatically stop the self-training procedure in the case of predictive performance degradation. This criterion has been evaluated on simple classification tasks, while here we consider it for MTR tasks. At each iteration, the out-of-bag error of the model is recorded. If an increase in the out-of-bag error is detected from one iteration to the next one, then the self-training procedure does not continue, the current model is discarded, and the model learned in the previous iteration is considered as the final model. Similarly to the procedure for automatic threshold selection (Section 3.4), the calculation of the out-of-bag error is based on the 'original' labeled examples (and not on examples labeled during the self-training procedure). Note that, in the case of *Airbag* stopping criteria, the first stopping criterion is also used in conjunction.

3.6. Computational complexity

The computational complexity of the self-training approach is the product of the number of iterations and the complexity of learning the base predictive model at each iteration. The theoretical upper bound for the number of iterations is equal to the number of unlabeled examples, assuming one unlabeled example is added to the training set per iteration. Note that this upper bound is typically never reached.

The complexity of learning the base model in this study, i.e., training random forests for predicting structured outputs, depends linearly on the number of decision trees, logarithmically on the number of descriptive attributes and $N \log N$ on the number of training instances [14]. Note that (1) the number of training instances is not constant in self-training, as it increases with each iteration when unlabeled examples are added to the training set and (2) the cost of the automatic threshold selection algorithm is linear in the number of training instances N .

4. Experimental design

In this section, we first describe the datasets used in the experimental evaluation. Next, we present the performance metrics, the evaluation procedure and the specific parameter settings of the algorithms. Finally, we state the main experimental questions of the study and describe the strategies used to answer them.

Table 4

Characteristics of the datasets. N : number of instances, D/C : number of descriptive attributes (discrete/continuous), T : number of target variables.

| Dataset (Reference) | N | D/C | T |
|---------------------------------|--------|-------|-----|
| RF1 [51] | 9125 | 0/64 | 8 |
| SCM1D [51] | 9803 | 0/280 | 16 |
| SCM20D [51] | 9803 | 0/61 | 16 |
| SIGMEA real [52] | 817 | 0/4 | 2 |
| SIGMEA simulated [52] | 10,368 | 2/9 | 2 |
| Soil quality [4] | 1944 | 0/142 | 3 |
| Solar flare-2 [53] | 1066 | 10/0 | 3 |
| Vegetation condition [6] | 16,967 | 1/39 | 7 |
| Water quality [54] | 1060 | 0/16 | 14 |

4.1. Data description

We use nine datasets with multiple continuous target variables to evaluate the predictive performance of the proposed methods. The datasets come from two different domains: environmental sciences and economy.

The majority of the data are from the area of environmental sciences. To begin with, the task in the *RF1* dataset is to predict river network flows 48 h ahead. Next, the *SIGMEA real* and *SIGMEA simulated* datasets consist of data about pollen dispersal rates on fields with genetically modified oilseed rape. Furthermore, the *Soil quality* and *Water quality* datasets concern habitat modelling of soil and water microorganisms, respectively. Next, the *Solar flare-2* dataset is about the number of solar flares of a certain class that occur over a 24 h period. Finally, the *Vegetation condition* dataset contains remote sensed vegetation data. From the domain of economy, we consider the *SCM1D* and *SCM20D* datasets, where the task is to predict the price of 16 products for the next day and their mean price over the next 20 days, respectively. We can observe (see Table 4) that the datasets vary in their size, number of attributes and number of target variables.

4.2. Experimental setup and evaluation procedure

The SSL method for MTR proposed in this study (CLUS-SSL) iteratively trains random forest ensembles for MTR. Hence, we compare the predictive performance of the self-training approach to the performance of a supervised random forest (CLUS-RF) – a baseline in the experimental evaluation. Additionally, we compare the proposed self-training approach to supervised multi-output support vector regression¹ (MSVR) [21,22]. A schematic representation of the self-training pipeline is presented in Fig. 1.

In all of the experiments, we construct random forests consisting of 100 predictive clustering trees for MTR. The trees are not pruned and the number of random features at each internal node is set to $\lfloor \log_2(D) + 1 \rfloor$, where D is the total number of features [19]. We used MSVR with a radial basis kernel while optimizing the C ($2^{-5}, 2^{-3}, \dots, 2^{13}$) and σ ($2^{-2}, 2^{-1}, \dots, 2^7$) parameters in a grid search procedure by performing 10 fold cross-validation on the labeled part of the data. The epsilon parameter was fixed ($\epsilon = 0.001$) [55].

The experimental evaluation uses different amounts of labeled data for both the supervised and SSL methods. To this end, we vary the proportion of labeled data used. The ratio of labeled (relative to unlabeled) data ranges in the following set: [1%, 5%, 10%, 15%, 20%, 30%, 50%].

For the CLUS-SSL algorithm, a threshold τ on the reliability score needs to be set. In order to estimate the potential performance of the various reliability scoring functions (independently

from the threshold), we consider 15 manually defined thresholds: $\tau = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.99\}$, resulting in 15 executions of the CLUS-SSL algorithm. From these executions, we report the best predictive performance. Obviously, we also evaluate performances obtained when the threshold is automatically set according to the four algorithms described in Section 3.4 (baselines: $T_{avg10\%}$, $T_{top10\%}$; our algorithms: $T_{OOBInitial}$ and T_{OOB}). For these four algorithms, we also investigate the effect of the Airbag stopping criterion.

The labeled data used for training the predictive models (both supervised and semi-supervised) are randomly selected from the available training data, while the remaining examples serve both as unlabeled data and as a test set. Namely, we temporarily remove their labels and provide the examples to the algorithm to serve as unlabeled data during training. At the end of self-training, the performance of the obtained model is evaluated on the test set composed of the unlabeled examples with the true labels restored (i.e., we consider a transductive-learning-setting). For fair comparison, supervised random forests are trained only on the labeled part of the data and evaluated on the same test set.

Note that the performance of self-training depends on the distribution of the initial labeled training data. For this reason, the performance of semi-supervised methods is, in general, known to be domain dependent [56]. To evaluate the robustness of our approach to the effect of possible different distributions of the labeled data, we repeated the random selection of labeled data (as described above) 10 times with different random initialization, to create 10 different labeled training sets. The predictive performance reported in the results is the average of the performance values obtained from the 10 runs.

We assess the predictive performance of the algorithms by using the *root-mean-square-error* (RMSE) defined as follows: $RMSE = \sqrt{\frac{1}{m} \sum_{i=1}^m RMSE_i^2}$, where m is the number of target variables and $RMSE_i$ is root mean square error of the i th target variable.

To investigate whether the observed differences in performance among the methods are statistically significant, we follow recommendations given by Demšar [57]. More specifically, to statistically compare the predictive performance of two methods over multiple datasets, we use the Wilcoxon signed ranks test [58], while for comparison of multiple methods, we use the corrected Friedman test and the post-hoc Nemenyi test [59]. We present the result from the Nemenyi post-hoc test with an average ranks diagram. The ranks are depicted on an axis, in such a manner that the best ranking algorithms are at the right-most side of the diagram. The algorithms that do not differ significantly (in performance) for a significance level of 0.05 are connected with a line.

All experiments were performed on a computer cluster which has 44 nodes and 984 central processing units (CPUs) in total: 9 nodes with 16 CPUs with an AMD Opteron processor at 800GHz on 64GB of RAM with the Fedora 24 operating system, 10 nodes with 24 CPUs with an AMD Opteron processor at 1900GHz on 128GB of RAM with the Fedora 24 operating system, and 25 nodes with 24 CPUs with an AMD Opteron processor at 1400GHz on 256GB of RAM with the Fedora 24 operating system. The CLUS-SSL and CLUS-RF algorithms are implemented in the Java programming language (version 1.6.), while MSVR is implemented in the C++ programming language.

4.3. Experimental questions

The principal goal of SSL is to improve the predictive performance of supervised learning by exploiting the information contained in unlabeled data. Therefore, the first question we want to answer in this study is: *Can self-training with any of the proposed reliability scores for MTR improve over the performance of supervised*

¹ <https://github.com/wjb19/mimo-svr>.

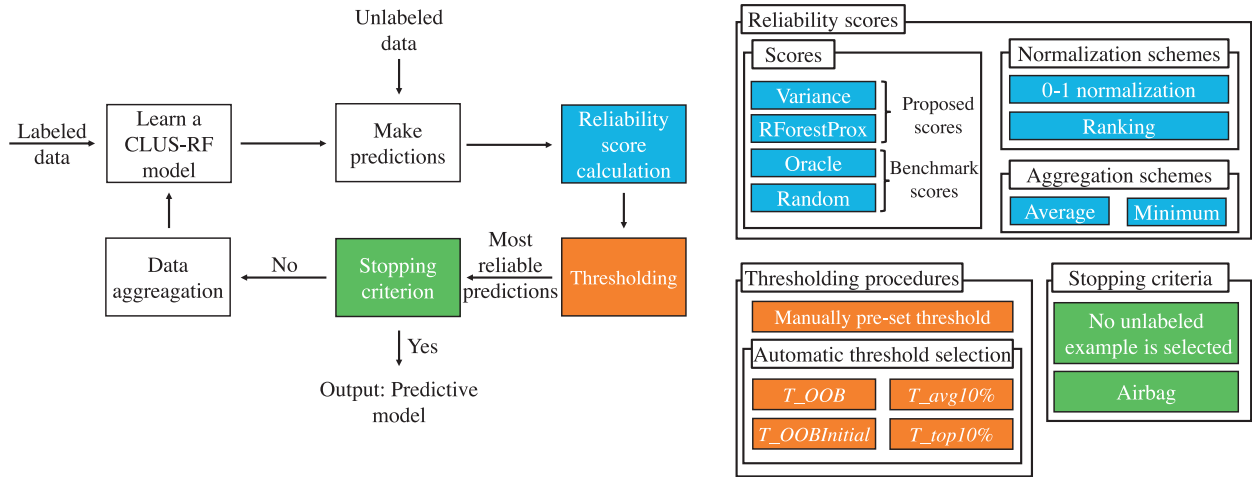


Fig. 1. The self-training approach to MTR. The left panel gives a schematic representation of the self-training approach, with all of its components. The right panel summarizes all of the possible choices we consider with respect to reliability scores, thresholding procedures, and stopping criteria.

learning? We address this question by comparing the predictive performance of the proposed method with the predictive performance of a supervised random forest.

If the answer to the first question is favourable, we are then interested in the following two questions: (1) *Is the performance increase due to the usage of reliability scores?*, and (2) *Which reliability score produces the largest improvement in performance?* To this end, we compare the proposed reliability scores (variance-based and ranking-based) to the benchmark scores (random score and oracle score). For question (1), we compare the performance of the proposed reliability scores with the performance obtained using random scores. If the proposed reliability scores have better predictive performance than random scores, then it is worthwhile to use a reliability scoring function and to look for an optimal function for this purpose. As for question (2), we compare the performance of the proposed reliability scores with the performance obtained using oracle scores. We expect that the oracle score performs best and we are optimistic that the performance of the proposed reliability scores is reasonably close to the performance of the oracle score.

Finally, we address the following question: *Is an automatic threshold selection algorithm able to exploit the advantage (if any) introduced by the reliability scores used in self-training?* To answer this question we evaluate the four different automatic threshold selection algorithms implemented in CLUS-SSL and compare them with the supervised random forest.

5. Results and discussion

In this section, we present the results obtained from the empirical evaluation of the various reliability scoring functions. We first present the evaluation of the methods' performance obtained using various portions of the data as training. Second, we compare the performance of SSL (CLUS-SSL) with the best performing reliability score with supervised learning (CLUS-RF and MSVR). Third, we illustrate a complete run of the CLUS-SSL method on a single dataset. Next, we compare the performance of CLUS-SSL with supervised learning (CLUS-RF and MSVR) when automatic threshold selection algorithms are used. Finally, we examine the per-target performance of the proposed method. In this section we only report summarized results: The complete and raw results are reported in [Appendix](#).

The first results we present are summarized in [Figs. 2](#) and [Fig. 3](#). [Fig. 2](#) gives the statistical evaluation of the performance of various methods using different portions of labeled data. The Ne-

menyi post-hoc analysis is performed for each portion of labeled data separately. [Fig. 3](#) summarizes the average rank diagrams and presents a global overview of the experimental evaluation. More specifically, this figure is obtained by plotting the multiple critical diagrams from the Nemenyi post-hoc test (from [Fig. 2](#)) jointly as follows. On the x-axis we depict the percentage of labeled data used, while on the y-axis we depict the average rank of a given method when applied to such data. These lines should not be treated as curves but as a parallel coordinate representation of the results of the statistical analysis. [Fig. 3](#) thus facilitates easier interpretation and understanding of the statistical analysis.

From the results, we can make several interesting observations. First of all, a major conclusion is that using unlabeled data (SSL) clearly improves the predictive performance of supervised random forests (CLUS-RF). Moreover, SSL using the benchmark oracle score with ranking-based aggregation (*OracleRankAvg* and *OracleRankMin*) is statistically significantly better than the supervised method in all of the settings with different percentages of labeled data. Furthermore, as hypothesized, the results of the proposed reliability scores are bounded by the results of the benchmark scores: *Random* is always ranked the lowest, while *OracleRankAvg* and *OracleRankMin* are always ranked the highest ([Fig. 3](#)). Finally, as expected, CLUS-RF and *Random* show similar behaviours.

A comparison of supervised and semi-supervised random forests to support vector regression (MSVR) reveals that both supervised and semi-supervised random forests outperform MSVR by a large margin. MSVR is better than CLUS-SSL and CLUS-RF only on 1 out of 9 datasets (Soil quality), while on the other 8 datasets, random forests are superior to MSVR across all percentages of labeled data ([Tables A.9–A.15](#)).

Next, we examine the contributions of the specific reliability scoring functions to the predictive performance ([Fig. 3](#)). All the proposed reliability scores (in all their variants) perform better than the *Random* score. This observation confirms that the used estimators of the reliability of predictions help to discern correct from erroneous predictions. Considering this and the fact that the oracle score with ranking-based averaging yields the best results, we conclude that instance selection in self-training for MTR strongly influences the predictive performance. In other words, if unlabeled instances are selected based on the reliability of their predictions during self-training, rather than selected randomly, better performance is achieved.

This finding is somewhat in disagreement with the empirical evaluation of self-training and co-training performed in the con-

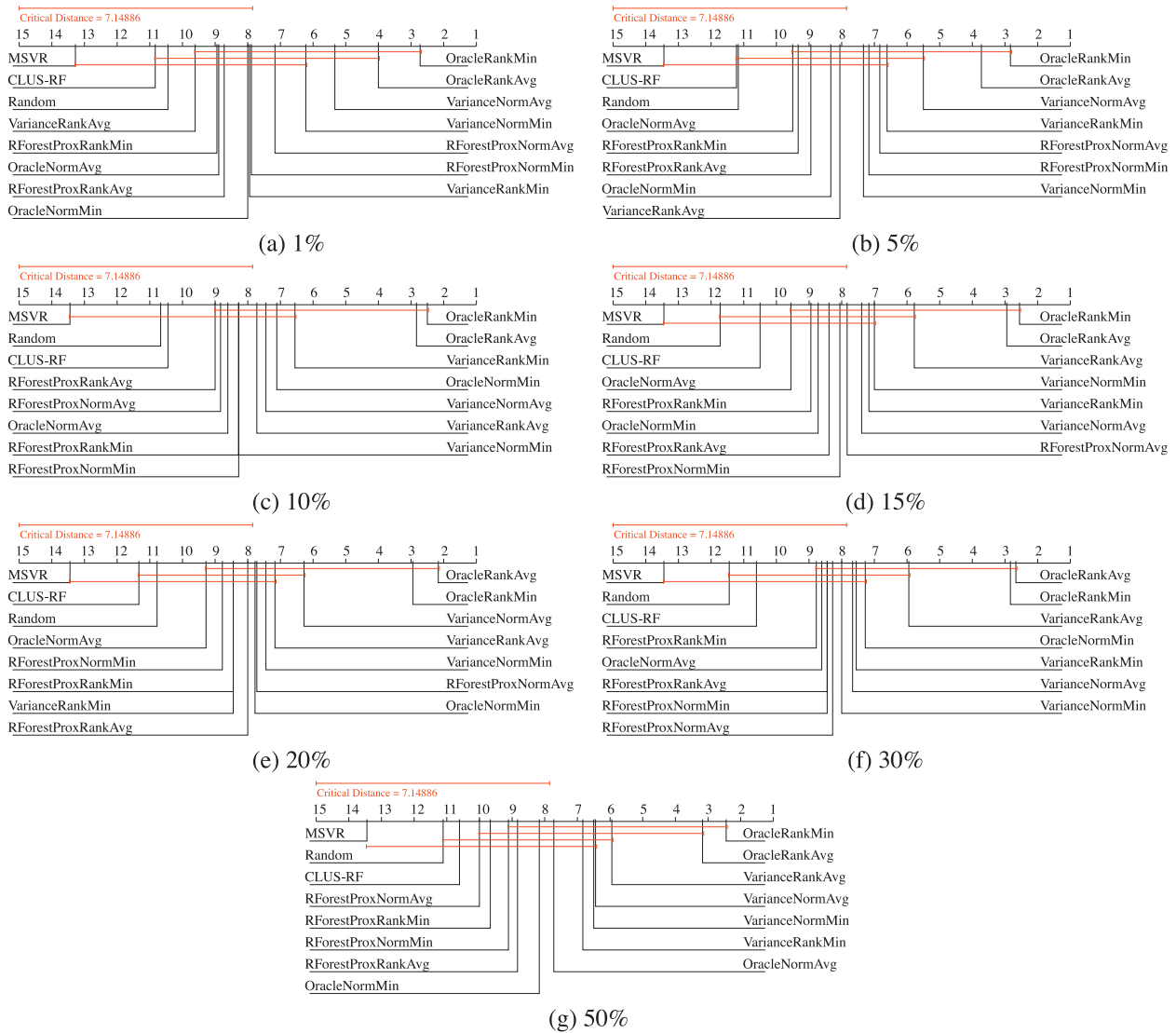


Fig. 2. Average ranks diagrams for the performance of the supervised algorithms (CLUS-RF and MSVR) and semi-supervised self-training with various reliability scores. The percentage of labeled data varies from 1% to 50%. Each graph represents two different kinds of information: the ranking among the algorithms (the algorithms positioned at the rightmost side of each graph are the best performing) and the statistical significance of the difference between pairs of algorithms (if their distance is less than the critical distance (at p -value = 0.05) there is no statistically significant difference between the two). The critical distance for all diagrams is 7.14886.

text of classification by Guo et al. [60]. They concluded that selecting the most confident unlabeled examples is not necessarily superior to random selection. In our study, random selection typically led to degradation of performance of the base model, and performed worse than the proposed reliability scores. Our results suggest that a proper reliability measure (i.e., one able to discern correct from erroneous predictions) will yield better performance of self-training than random instance selection.

The better ranking of SSL methods with the proposed reliability scores as compared to the supervised method is consistent across the different percentages of labeled data. Reliability scores based on the variance of votes of an ensemble, in most of the cases, perform better than reliability scores based on random forest proximities. Among the variance-based reliability scores, *VarianceNormAvg* is clearly one of the best performing. Moreover, the difference in performance between the two best scoring functions (*OracleRankAvg* and *OracleRankMin*) is not statistically significant.

The proposed reliability scores use two aggregation schemes (average and minimum), and two normalization schemes (min-max and ranking-based normalization) for the per-target scores.

The results suggest that the two aggregation schemes do not show significant differences in performance. On the other hand, between the two normalization approaches, min-max normalization is favourable, since *VarianceNorm* and *RForestProxNorm* are (in most of cases) ranked better than their counterparts based on ranking (*VarianceRank* and *RForestProxRank*). However, this is not the case with the benchmark oracle score: *OracleRank* has better predictive performance than *OracleNorm*. We find this to be an artefact of the (skewed) distribution of errors on unlabeled examples. We further illustrate this phenomenon in Fig. 4 for the *RF1* dataset (similar observations are made also for other datasets). In particular, we report the frequencies of the RMSE errors discretized into bins in Fig. 4a and b. We see that the distributions of per-target errors of unlabeled examples are highly skewed, where few examples have much higher errors than the rest of examples.

Next, recall that the oracle scores (*OracleRank* and *OracleNorm*) are calculated by using the actual errors on unlabeled examples. The *OracleNorm* reliability score averages (or takes the minimum of) the normalized per-target errors, therefore, such distributions of per-target errors yield reliability scores biased towards high re-

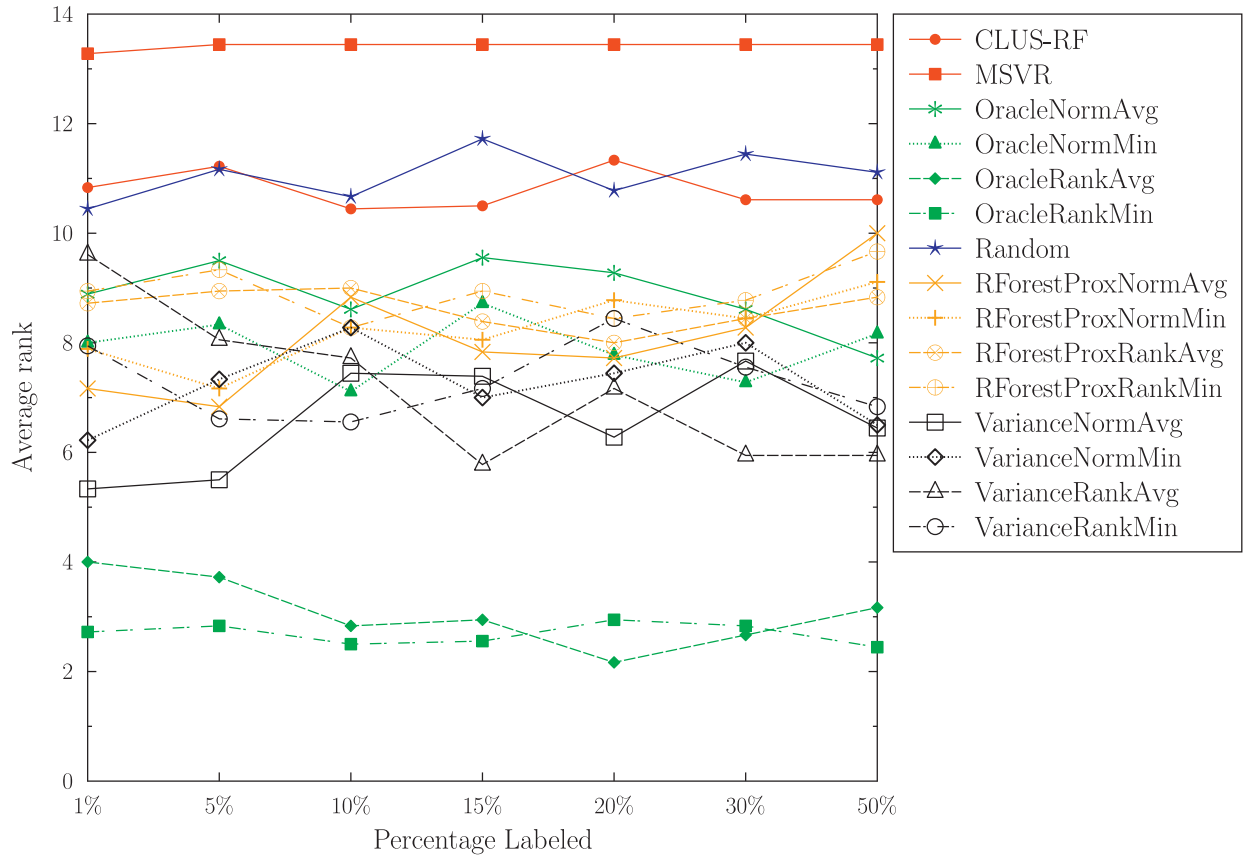


Fig. 3. Average ranks for different percentages of labeled data (from 1% to 50%) of supervised methods (red lines), self-training with the proposed reliability scores (black and orange lines) and self-training with benchmark scores (blue and green lines). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

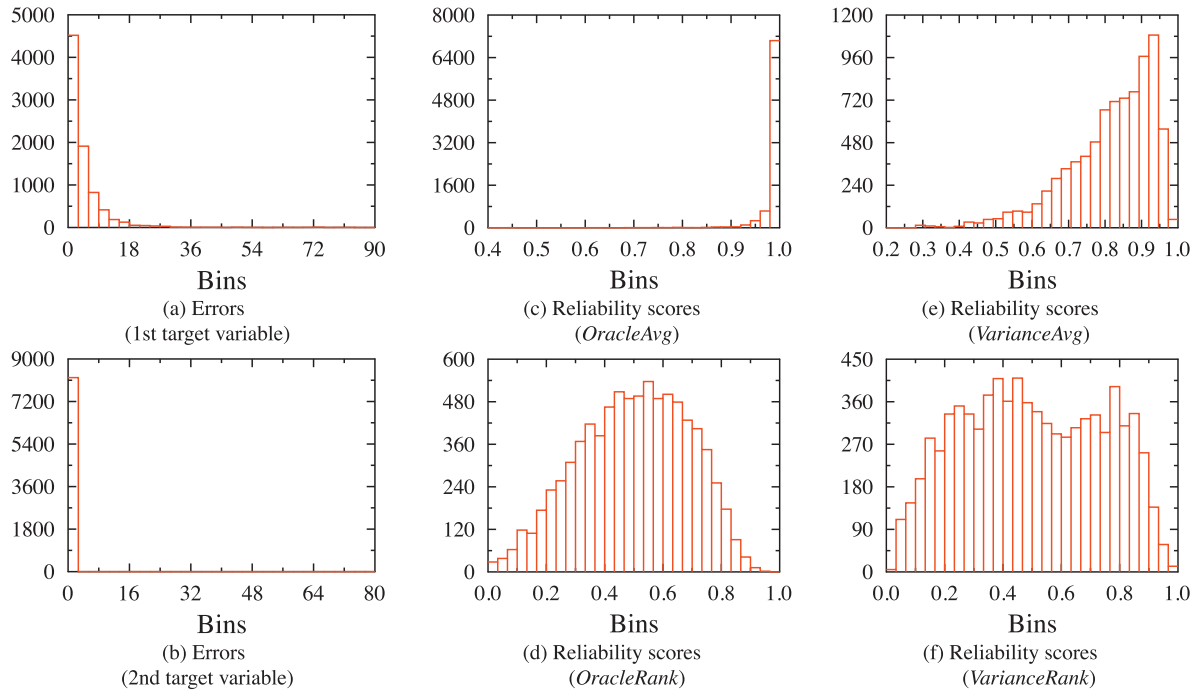


Fig. 4. The distributions of per-target errors (discretized into bins) made on unlabeled examples after the initial iteration of self-training are very skewed for the target variables of the RF1 dataset when 10% of the data are labeled. These are depicted for the first two target variables (a, b), similar distributions are observed also for the other six target variables of the RF1 dataset. The reliability scores (discretized into bins) for unlabeled examples calculated by averaging normalized per-target errors are biased towards high reliability (c, e), while the scores calculated by averaging ranks of per-target errors are normally distributed (d, f).

Table 5

Results of the Wilcoxon signed-rank test applied to the performance of self-training with *VarianceNormAvg* reliability score and the performance of supervised random forest (CLUS-RF) on the 9 datasets considered in this study. In bold, we report significant *p*-values (< 0.05).

| % Labeled | 1% | 5% | 10% | 15% | 20% | 30% | 50% |
|-----------------|--|--|--|--------------------------------------|--|--|--|
| <i>p</i> -value | 1.5×10^{-5} | 1.7×10^{-8} | 2.2×10^{-6} | 4×10^{-8} | 1.7×10^{-6} | 4.2×10^{-5} | 4.8×10^{-6} |

liability scores (i.e., the majority of unlabeled examples are given a reliability score close to one, Fig. 4c). In other words, during the initial iterations of self-training, erroneously predicted unlabeled examples are easily deemed reliable, and are thus likely to be added to the training set. A prediction error made in early iterations of self-training can propagate itself in the next iterations, leading to a degradation of the performance.

Regardless of the relatively poor performance of self-training with the *OracleNorm* score, the reliability scores given by *OracleNorm* are still “oracle” scores, i.e., from two unlabeled examples, the one with the smaller error will always get the higher reliability score. It can be expected that fine tuning the reliability threshold in the region of high reliability (from 0.9 to 0.99) would likely result in improved performance of *OracleNorm*. The *OracleRank* reliability score does not suffer from this problem, since the ranks of the per-target errors are averaged (and not the errors themselves), which yields normally distributed (i.e., unbiased) scores (Fig. 4d).

An additional observation we can make from Fig. 4e and f is that the frequency distribution of the (Norm and Rank) variance reliability scores conforms to the frequency distribution of the corresponding (Norm and Rank) oracle reliability scores. This is an indication that our reliability scores, in this particular dataset, approximate the real distributions of the errors and, thus, identify the best predictions to be considered as reliable.

Based on this analysis, we select the best performing aggregation scheme for each of the different types of reliability scores (Variance, Random forest proximities and Oracle) as follows: *VarianceNormAvg*, *RForestProxNormMin* and *OracleRankMin*. To better illustrate the difference in the performance, we compare the performance of these against the performance of supervised random forests (CLUS-RF) and random scores (*Random*). The results of the statistical analysis (Friedman test and Nemenyi post-hoc analysis) are given in Fig. A1. The findings from this simplified analysis concur with the findings above.

As noted before, among the proposed reliability scores, one of the best performing ones is *VarianceNormAvg*. To inspect if the differences in performance at the various percentages of labeled data between this reliability score and supervised learning (CLUS-RF) are statistically significant, we use the Wilcoxon signed-rank test. The results of the Wilcoxon test are given in Table 5. They show that self-training with *VarianceNormAvg* is statistically significantly better than CLUS-RF (*p*-value < 0.05) for all percentages of labeled data.

In order to demonstrate the possible effects of the value of the reliability threshold on the performance of self-training, we present an example of results obtained on the RF1 dataset by applying the different thresholds (Fig. 5). These results show that selecting a too permissive threshold (i.e., smaller value for the threshold) can allow wrongly predicted examples to enter into the training set, leading to a degradation of the performance. For example, if a threshold smaller than 0.85 on the *VarianceNormAvg* reliability score at 1% of labeled data is chosen, a performance worse than with the random instance selection can be observed (Fig. 5). On the other hand, a too stringent threshold can prevent self-training to learn from unlabeled data, i.e., self-training will not improve the performance of supervised learning. For example, if a threshold greater than 0.8 is set for the *OracleRankAvg* reliabil-

Table 6

Average number of iterations (across the 9 dataset considered in this study) of the self-training with different algorithms for automatic reliability threshold selection and manual threshold selection (*VarianceNormAvg*).

| Method | Percentage labeled | | | | | | |
|------------------------------|--------------------|------|------|------|------|------|------|
| | 1% | 5% | 10% | 15% | 20% | 30% | 50% |
| VarianceNormAvg | 15.3 | 37.4 | 21.0 | 64.5 | 31.2 | 34.9 | 45.2 |
| <i>T_OOB</i> | 22.3 | 42.3 | 36.8 | 35.2 | 34.7 | 34.9 | 31.1 |
| <i>T_OOB</i> + Airbag | 6.6 | 7.2 | 6.7 | 7.6 | 8.3 | 8.5 | 7.9 |
| <i>T_OOBInitial</i> | 21.2 | 67.5 | 49.6 | 51.3 | 52.1 | 58.3 | 40.0 |
| <i>T_OOBInitial</i> + Airbag | 3.4 | 3.5 | 3.5 | 3.2 | 3.2 | 3.1 | 2.8 |
| <i>Ttop10%</i> | 11.0 | 11.0 | 11.0 | 11.0 | 11.0 | 11.0 | 11.0 |
| <i>Ttop10%</i> + Airbag | 3.8 | 3.3 | 3.2 | 3.1 | 3.4 | 2.9 | 2.7 |
| <i>Tavg10%</i> | 77.5 | 90.0 | 82.0 | 73.8 | 69.3 | 63.4 | 40.0 |
| <i>Tavg10%</i> + Airbag | 4.1 | 2.8 | 2.7 | 2.5 | 2.7 | 2.4 | 2.4 |

ity score at 1% or 5% of labeled data on the RF1 dataset, none, or very few, of the unlabeled examples enter the training set, meaning that we miss the opportunity to improve performance by using unlabeled data (Fig. 5).

After this discussion, it is clear that the reliability threshold selection is critical to the success of the self-training approach to SSL. We next investigate the effectiveness of the algorithms for automatic threshold selection that we have proposed. Fig. 6, allows us to compare results obtained by CLUS-RF and MSVR with results obtained when the four automatic threshold selection algorithms (*Tavg10%*, *Ttop10%*, *T_OOBInitial* and *T_OOB*) are used in CLUS-SSL (with the *VarianceNormAvg* reliability score). The results confirm that automatic threshold selection is not an easy task in self-training and that choosing the best algorithm for this task is crucial to profit from the semi-supervised learning framework. In the specific case we consider, we can see that the algorithm proposed by Tanha et al. [50] (i.e., *Tavg10%*) in most of cases, puts semi-supervised learning at a disadvantage. This is not the case for the algorithms we propose in this paper (*T_OOB* and *T_OOBInitial*). These algorithms materialize the SSL advantage by exploiting the out-of-bag error of the random forest (even if, according to the Wilcoxon signed rank test, there is no clear statistical evidence that *T_OOB* and *T_OOBInitial* outperform CLUS-RF). Moreover, adapting the threshold at each iteration is generally better than choosing it only once after the initial iteration (*T_OOB* vs *T_OOBInitial*).

A good indication of how close the automatically identified thresholds are to the optimal ones is provided by the number of iterations. As we can see from Table 6, the algorithms *T_OOB* and *T_OOBInitial* lead to convergence in a number of iterations which is similar to the number of iterations when the threshold is ‘hand-picked’. This is not true for the other algorithms for the automatic identification of the threshold.

When we consider automatic threshold selection in combination with the Airbag stopping criterion (Fig. 6), we can see that we are able to considerably improve performance and outperform, also with a statistically significant difference, CLUS-RF. This is especially true for *T_OOBInitial* when the percentage of labeled examples is greater than 10% (see Table 8). When we have a small percentage of labeled examples, the simple algorithm *Tavg10%* outperforms *T_OOBInitial*. This difference between the two algorithms for automatic threshold selection was expected. In fact, the out-of-bag error is computed only on labeled examples and this can lead to

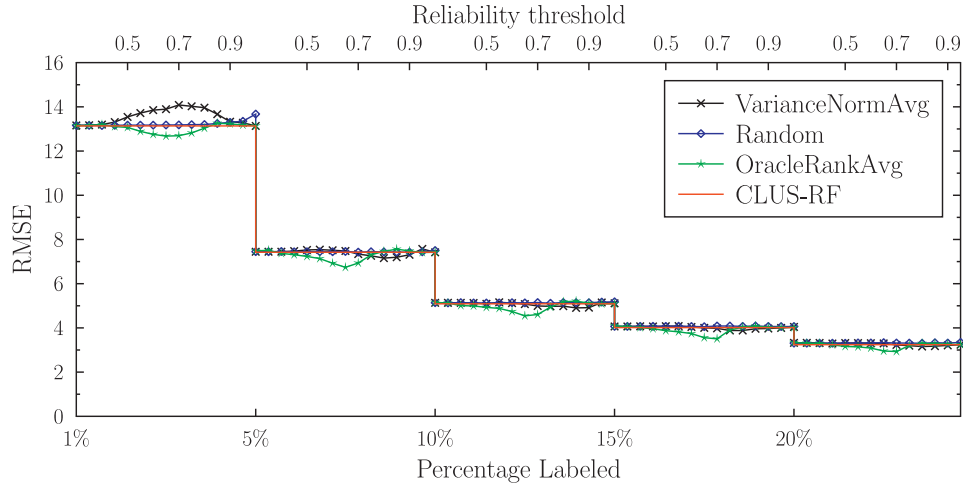


Fig. 5. A comparison of the predictive performance of random forests (CLUS-RF) and self-training with different reliability scores (*VarianceNormAvg*, *Random* and *OracleRankAvg*) on the RF1 dataset illustrates the possible effects of threshold selection: A too permissive threshold for the *VarianceNormAvg* reliability score leads to worse performance than random instance selection, while a too strict threshold for the *OracleRankAvg* reliability score does not allow self-training to benefit from the unlabeled data.

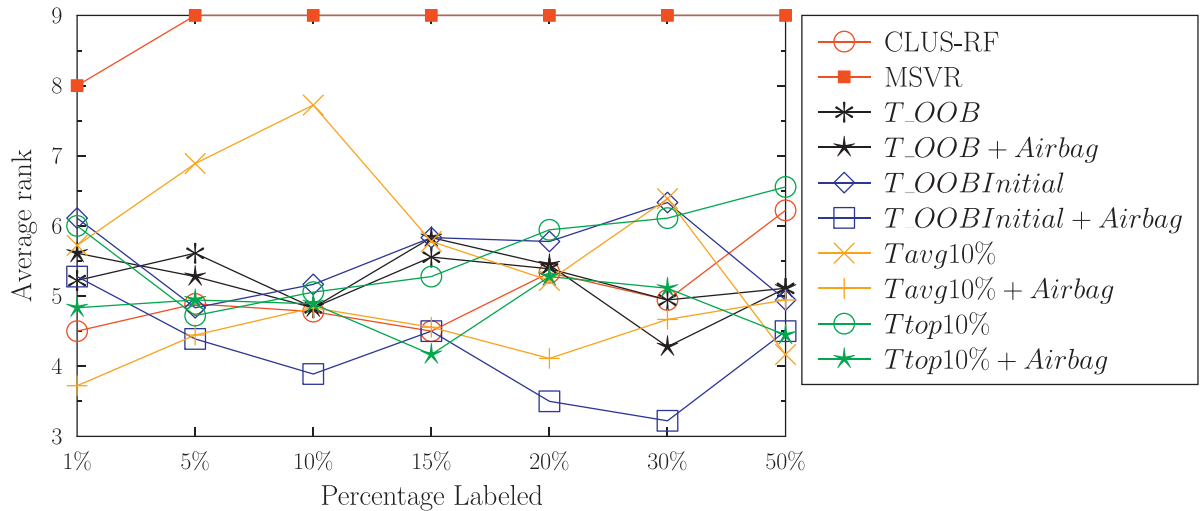


Fig. 6. Average ranks for different percentages of labeled data (from 1% to 50%) of supervised algorithms (red lines), and self-training *VarianceNormAvg* with the baseline automatic thresholding algorithms (orange and green lines) and self-training *VarianceNormAvg* with the proposed automatic thresholding algorithms (black and blue lines). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

wrong decisions about thresholds when we have very few labeled examples. Moreover, *Tavg10%* outperforms CLUS-RF when the percentage of labeled examples is very small ($= 1\%$, see Table 8 and Fig. 6). We notice from Table 6 that the Airbag stopping criteria strongly reduces the average number of iterations, confirming that a large number of iterations, without an optimal threshold, is not beneficial.

Since the self-training method iteratively re-trains the base method, obviously, potential predictive performance of self-training comes with a cost of increased learning time. Table 7 reports learning times of CLUS-RF, MSVR and self-training. We can observe that learning time of self-training can be up to several hundred times larger than the one of CLUS-RF (e.g., for *Tavg10%* or *T_OOBInitial*). However, as we previously stated, the Airbag stopping criterion is not only beneficial for predictive performance, but it also greatly reduces the number of iterations of self-training. Since the learning time of self-training is strongly correlated with the number of iterations (Tables 6 and 7), the learning time of self-training with the Airbag stopping criterion is greatly reduced if compared to self-training without this stop-

ping criterion. The learning time of the best performing methods (i.e., *T_OOBInitial+Airbag* and *Tavg10%+Airbag*) is typically only 3–5 times larger than the learning time of CLUS-RF. The MSVR algorithm typically runs faster than CLUS-RF or self-training. This is in line with the time complexity analysis reported in Section 3.6, where we point out that the time complexity of our approach is $N \log N$ in the number of training instances, logarithmic in the number of attributes and linear in the number of iterations, whereas the time complexity of MSVR is linear in the number of training instances, linear in the number of descriptive attributes and linear in the number of iterations² [21]. However, much more time is used in MSVR for parameter optimization, especially as the amount of labeled data increases. Note that the predictive performance of MSVR can vary extremely depending on parameter values; therefore, it is advisable to perform parameter optimization.

Finally, Fig. 7 presents a different perspective on the results, highly relevant to MTR problems, by illustrating the per-target

² MSVR performs a quasi-Newton approach in which each iteration better approximates the model.

Table 7

Average learning time in seconds (across the 9 dataset considered in this study) of supervised methods (CLUS-RF and MSVR), self-training with different algorithms for automatic reliability threshold selection and manual threshold selection (*VarianceNormAvg*). For the MSVR algorithm, learning time spent for parameter optimization is presented in brackets.

| Method | Percentage labeled | | | | | | |
|-------------------------------|--------------------|-------------|-------------|-------------|-------------|--------------|--------------|
| | 1% | 5% | 10% | 15% | 20% | 30% | 50% |
| CLUS-RF | 2.4 | 3.5 | 4.5 | 5.8 | 6.6 | 9.5 | 13.9 |
| MSVR | 0.01(256.5) | 0.1(1437.4) | 0.2(1864.2) | 0.5(3113.1) | 1.7(6972.1) | 2.4(13354.4) | 8.5(31843.7) |
| VarianceNormAvg | 116.9 | 872.8 | 464.9 | 667.9 | 625.9 | 1133.6 | 1989.6 |
| <i>T_OOB</i> | 542.7 | 1205 | 689.2 | 671.5 | 649.5 | 781.7 | 867.5 |
| <i>T_OOB</i> + Airbag | 149.9 | 163.6 | 72.5 | 88.1 | 98.2 | 128 | 167.5 |
| <i>T_OOB</i> Initial | 588.4 | 2118.1 | 1281.6 | 1518 | 1548 | 1704.3 | 1001.8 |
| <i>T_OOB</i> Initial + Airbag | 69.5 | 32.7 | 24.2 | 20.3 | 22.3 | 24.9 | 40.4 |
| <i>Ttop10%</i> | 189.8 | 195.8 | 205.8 | 201.1 | 207.5 | 225.3 | 238.3 |
| <i>Ttop10%</i> + Airbag | 10.3 | 12.2 | 16.7 | 18.5 | 25.2 | 26.3 | 37.7 |
| <i>Tavg10%</i> | 1871.3 | 2365.9 | 2104.2 | 1847.8 | 1745.4 | 1577.3 | 1013.3 |
| <i>Tavg10%</i> + Airbag | 10.6 | 8.6 | 10.7 | 13.4 | 16.8 | 21.1 | 37.2 |

Table 8

P-values of the Wilcoxon signed-rank test applied to the performances (RMSE) of a supervised random forest (CLUS-RF) and self-training with algorithms for automatic threshold selection (*T_OOB*Initial + Airbag and *Tavg10%* + Airbag) on the 9 datasets considered in this study. In bold, we report significant *p*-values (<0.05). The ‘-’ sign denotes that CLUS-RF is better, while ‘+’ that self-training is better.

| % Labeled | 1% | 5% | 10% | 15% | 20% | 30% | 50% |
|-------------------------------|----------|-----------|-----------|------------------|-----------------|------------------|------------------|
| <i>T_OOB</i> Initial + Airbag | 0.41 (+) | 0.162 (-) | 0.251 (-) | 0.034 (+) | 0.01 (+) | 0.008 (+) | 0.012 (+) |
| <i>Tavg10%</i> + Airbag | 0.13 (+) | 0.075 (-) | 0.286 (-) | 0.33 (+) | 0.32 (+) | 0.186 (-) | 0.015 (+) |

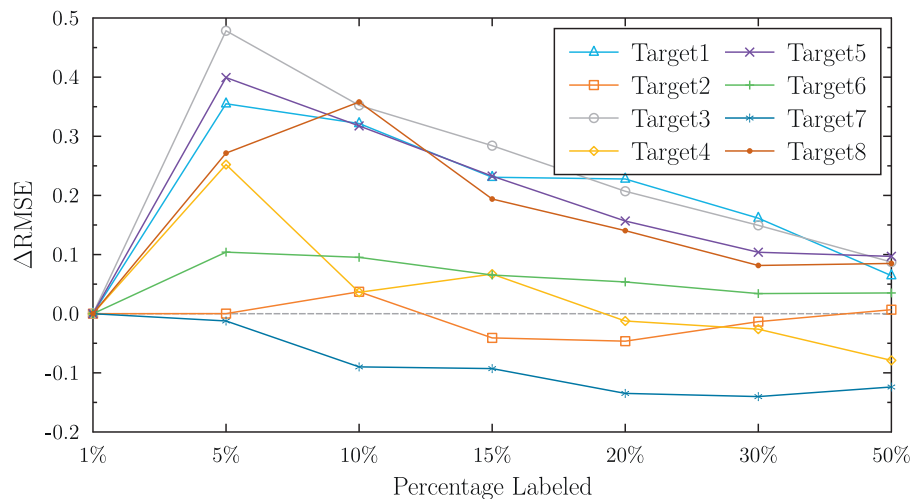


Fig. 7. Analysis of per-target performance for the RF1 dataset, in terms of difference in performance (Δ RMSE) between CLUS-RF and self-training with the *VarianceNormAvg* reliability score. Positive values suggest that self-training is better, while negative that CLUS-RF is better. Zero means that there is no difference in performance. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

RMSE performance of self-training. More specifically, it presents the per-target RMSE improvements of self-training with the *VarianceNormAvg* reliability score over supervised random forest (CLUS-RF). These results show that the improvement provided by the SSL setting is not necessarily uniform across all of the different target variables. Self-training improves performance over CLUS-RF for all of the targets when the percentage of labeled examples is relatively small. Otherwise it improves the performance for most of the target variables, but can degrade the performance for some target variables (e.g., target no. 7 for > 5% of labeled data).

6. Conclusions

In this paper, we address the task of semi-supervised learning for multi-target regression – a type of structured output prediction, where the goal is to simultaneously predict multiple continuous variables. To the best of our knowledge, general purpose semi-

supervised methods dealing with this task do not exist thus far. We propose a self-training approach to semi-supervised learning by using a random forest of predictive clustering trees for multi-target regression. In the proposed approach, a model uses its own most reliable predictions in an iterative fashion. Therefore, a proper measure for the reliability of predictions is of crucial importance.

We propose two reliability scoring functions for multi-target predictions, two aggregation schemes for merging per-target scores into a global score and two normalization techniques, resulting in a total of eight distinct scores. The proposed reliability scoring functions are based on the mechanisms provided by ensemble learning: the variance of the votes of an ensemble and the estimate of errors of unlabeled examples by using out-of-bag labeled examples in their random forest proximities.

Our empirical evaluation conducted on real datasets for multi-target regression, shows that self-training with any of the eight proposed reliability measures is able to consistently improve

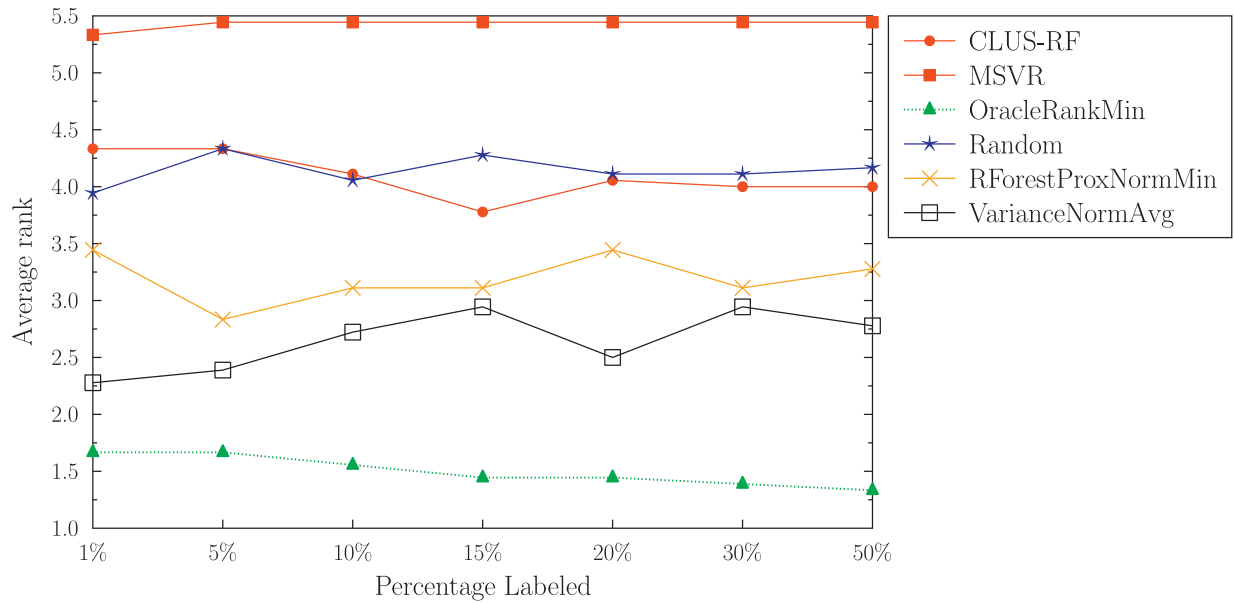


Fig. A1. Average ranks for different percentages of labeled data (from 1% to 50%) of supervised methods (red lines), self-training with the best reliability score based on variance (black), the best score based on random forest proximities (orange), the best benchmark score (green), and random score (blue). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

over supervised random forests and outperform supervised multi-output support vector regression. Among the two strategies for merging per-target scores, the averaging of the scores showed to be more favourable than the averaging of their ranks. The averaged variance-based reliability score (*VarianceNormAvg*), which is based the variance of votes of an ensemble, is the best performing reliability score among the proposed reliability scores. Random selection during self-training is clearly inferior to the use of reliability scores: Better performance is achieved if unlabeled examples are selected on the basis of reliability scores, than if they are selected randomly.

Despite this potential superiority of self-training over supervised approaches when learning random forests, choosing the best threshold for the reliability of predictions still remains an issue: the best threshold depends on the specific domain and can vary from iteration to iteration. In this respect, we proved that automatic identification of the threshold is possible but, depending on the amount of labeled examples we have, different algorithms should be used. The algorithm automatic threshold selection based on the exploitation of out-of-bag errors ($T_{OOBinitial}$) is recommended when more than 5% of labeled (as compared to unlabeled) data is available. For very small amounts of labeled data ($< 5\%$) estimates of out-of-bag errors are not reliable; therefore, a simple solution is more suitable: using the average score of the top 10% of most reliable predictions. The use of the Airbag stopping criteria during the automatic threshold selection proved to be beneficial in all cases.

There are several directions to extend our work in the future. A first possible direction concerns handling per-target reliability scores separately: Losses of accuracy related only to individual target variables could be avoided by optimizing thresholds specifi-

cally for each target. This would require the development of methods capable of dealing with partially labeled data. Dealing with such type of data has not received much attention in the semi-supervised learning community, but it is particularly relevant for the task of structured output prediction. We would like to perform a more extensive evaluation on more MTR datasets and achieve better understanding of when SSL (i.e., at which datasets) improves performance (as compared to SSL). Moreover, we would like to extend our self-training approach to other tasks of structured output prediction, such as (hierarchical) multi-label classification and predicting time-series, and evaluate its performance on practically relevant tasks of these types, e.g., gene function prediction.

Acknowledgment

We acknowledge the financial support of the Slovenian Research Agency, via the grant P2-0103 and a young researcher grant to the first author, as well as the European Commission, via the grants ICT-2013-612944 MAESTRA and ICT-2013-604102 HBP.

Appendix

In this appendix, we report the RMSE results according to which the figures and tables reported in this paper have been drawn. In particular, Table A.9 shows results obtained with 1% of labeled data, Table A.10 shows results obtained with 5% of labeled data, Table A.11 shows results obtained with 10% of labeled data, Table A.12 shows results obtained with 15% of labeled data, Table A.13 shows results obtained with 20% of labeled data, Table A.14 shows results obtained with 30% of labeled data, and Table A.15 shows results obtained with 50% of labeled data.

Table A.9

RMSE on the 9 considered datasets obtained with 1% of labeled data by supervised methods (CLUS-RF and MSVR), self-training with the proposed reliability scores (VarianceNormAvg, VarianceNormMin, VarianceRankAvg, VarianceRankMin, RForestProxNormAvg, RForestProxNormMin, RForestProxRankAvg and RForestProxRankMin), and self-training with benchmark scores (Random, OracleNormAvg, OracleNormMin, OracleRankAvg and OracleRankMin). The lower sub-table presents the performance of self-training with the VarianceNormAvg reliability score and several procedures for automatic selection of a threshold for the reliability score (T_{OOB} , $T_{OOBInitial}$, $T_{top10\%}$ and $T_{avg10\%}$), with or without the Airbag stopping criteria. The best result for each column of each sub-table is marked in bold. The last column presents the average rank of the methods according to their performance on the 9 datasets (calculated separately for each sub-table).

| Method | Dataset | | | | | | | | | Average rank |
|----------------------------------|--------------|--------------|--------------|------------------|---------------|------------------|----------------|----------------|---------------|----------------------|
| | Sigma Real | Sigma Sim. | Soil Quality | Solar Flare-2 | Water Quality | Vegetation Cond. | SCM20D | SCM1D | RF1 | |
| CLUS-RF | 6.422 | 0.062 | 0.547 | 45909.739 | 1.372 | 2.871 | 206.99 | 175.969 | 13.136 | 10.833 ± 4.430 |
| MSVR | 6.555 | 0.185 | 0.217 | 47440.970 | 1.354 | 3.105 | 230.765 | 265.669 | 22.911 | 13.278 ± 4.631 |
| VarianceNormAvg | 6.4 | 0.036 | 0.536 | 45886.432 | 1.352 | 2.866 | 206.832 | 175.924 | 13.136 | 5.333 ± 4.077 |
| VarianceNormMin | 6.408 | 0.036 | 0.538 | 45913.693 | 1.354 | 2.864 | 206.99 | 175.969 | 12.998 | 6.222 ± 3.465 |
| VarianceRankAvg | 6.406 | 0.037 | 0.541 | 46016.187 | 1.348 | 2.868 | 207.276 | 176.801 | 13.048 | 9.611 ± 3.018 |
| VarianceRankMin | 6.388 | 0.039 | 0.538 | 45962.634 | 1.346 | 2.869 | 208.656 | 176.555 | 12.968 | 7.944 ± 3.321 |
| RForestProxNormAvg | 6.408 | 0.047 | 0.54 | 45870.267 | 1.341 | 2.866 | 206.99 | 175.969 | 13.127 | 7.167 ± 2.681 |
| RForestProxNormMin | 6.405 | 0.048 | 0.54 | 45909.739 | 1.343 | 2.866 | 206.99 | 175.969 | 13.156 | 7.889 ± 3.286 |
| RForestProxRankAvg | 6.405 | 0.047 | 0.536 | 45934.234 | 1.34 | 2.872 | 207.323 | 176.772 | 13.115 | 8.722 ± 3.759 |
| RForestProxRankMin | 6.416 | 0.046 | 0.539 | 45850.644 | 1.346 | 2.871 | 210.392 | 176.134 | 13.064 | 8.944 ± 3.283 |
| Random | 6.38 | 0.055 | 0.54 | 46262.372 | 1.346 | 2.866 | 207.383 | 176.883 | 13.155 | 10.444 ± 3.787 |
| OracleNormAvg | 6.421 | 0.04 | 0.543 | 46224.958 | 1.341 | 2.864 | 207.229 | 176.619 | 13.061 | 8.889 ± 3.765 |
| OracleNormMin | 6.421 | 0.038 | 0.541 | 46000.04 | 1.336 | 2.86 | 207.242 | 176.626 | 13.069 | 8 ± 4.493 |
| OracleRankAvg | 6.375 | 0.03 | 0.54 | 45765.407 | 1.338 | 2.867 | 206.99 | 175.969 | 12.676 | 4 ± 3.260 |
| OracleRankMin | 6.375 | 0.032 | 0.537 | 45694.472 | 1.337 | 2.864 | 206.99 | 175.969 | 12.878 | 2.722 ± 1.325 |
| CLUS-RF | 6.422 | 0.062 | 0.547 | 45909.739 | 1.372 | 2.871 | 206.99 | 175.969 | 13.136 | 4.5 ± 4.031 |
| MSVR | 6.555 | 0.185 | 0.217 | 47440.97 | 1.354 | 3.105 | 230.765 | 265.669 | 22.911 | 8 ± 3.708 |
| T_{OOB} | 6.422 | 0.036 | 0.545 | 46251.345 | 1.368 | 2.93 | 207.341 | 178.391 | 13.427 | 5.222 ± 1.970 |
| $T_{OOB} + \text{Airbag}$ | 6.422 | 0.046 | 0.545 | 46251.345 | 1.368 | 2.905 | 207.341 | 178.391 | 13.461 | 5.611 ± 1.387 |
| $T_{OOBInitial}$ | 6.422 | 0.04 | 0.545 | 46251.345 | 1.368 | 3.037 | 207.332 | 178.662 | 13.927 | 6.111 ± 2.205 |
| $T_{OOBInitial} + \text{Airbag}$ | 6.422 | 0.046 | 0.545 | 46251.345 | 1.368 | 2.912 | 207.331 | 178.526 | 13.332 | 5.278 ± 1.716 |
| $T_{top10\%}$ | 6.432 | 0.044 | 0.542 | 46337.764 | 1.351 | 3.027 | 216.671 | 189.056 | 13.672 | 6 ± 2.872 |
| $T_{top10\%} + \text{Airbag}$ | 6.439 | 0.059 | 0.544 | 46114.348 | 1.366 | 2.867 | 209.574 | 177.558 | 13.204 | 4.833 ± 2.598 |
| $T_{avg10\%}$ | 6.408 | 0.037 | 0.543 | 46433.652 | 1.355 | 3.109 | 221.755 | 207.888 | 13.403 | 5.722 ± 3.528 |
| $T_{avg10\%} + \text{Airbag}$ | 6.422 | 0.059 | 0.543 | 45926.363 | 1.363 | 2.875 | 209.183 | 177.029 | 13.134 | 3.722 ± 2.063 |

Table A.10

RMSE on the 9 considered datasets obtained with 5% of labeled data by supervised methods (CLUS-RF and MSVR), self-training with the proposed reliability scores (VarianceNormAvg, VarianceNormMin, VarianceRankAvg, VarianceRankMin, RForestProxNormAvg, RForestProxNormMin, RForestProxRankAvg and RForestProxRankMin), and self-training with benchmark scores (Random, OracleNormAvg, OracleNormMin, OracleRankAvg and OracleRankMin). The lower sub-table presents the performance of self-training with the VarianceNormAvg reliability score and several procedures for automatic selection of a threshold for the reliability score (T_{OOB} , $T_{OOBInitial}$, $T_{top10\%}$ and $T_{avg10\%}$), with or without the Airbag stopping criteria. The best result for each column of each sub-table is marked in bold. The last column presents the average rank of the methods according to their performance on the 9 datasets (calculated separately for each sub-table).

| Method | Dataset | | | | | | | | | Average rank |
|----------------------------------|--------------|--------------|--------------|------------------|---------------|------------------|----------------|----------------|--------------|----------------------|
| | Sigma Real | Sigma Sim. | Soil Quality | Solar Flare-2 | Water Quality | Vegetation Cond. | SCM20D | SCM1D | RF1 | |
| CLUS-RF | 5.431 | 0.019 | 0.524 | 41384.376 | 1.262 | 2.678 | 177.182 | 143.124 | 7.422 | 11.222 ± 3.743 |
| MSVR | 6.337 | 0.816 | 0.168 | 45561.610 | 1.304 | 2.988 | 230.311 | 229.473 | 23.985 | 13.444 ± 4.667 |
| VarianceNormAvg | 5.241 | 0.018 | 0.513 | 41384.376 | 1.253 | 2.673 | 177.153 | 143.124 | 7.157 | 5.5 ± 3.873 |
| VarianceNormMin | 5.305 | 0.018 | 0.511 | 41384.376 | 1.255 | 2.676 | 177.182 | 143.079 | 7.359 | 7.333 ± 2.739 |
| VarianceRankAvg | 5.239 | 0.017 | 0.513 | 41334.819 | 1.254 | 2.679 | 178.049 | 144.487 | 7.177 | 8.056 ± 4.482 |
| VarianceRankMin | 5.281 | 0.017 | 0.51 | 41239.703 | 1.256 | 2.676 | 177.207 | 143.582 | 7.269 | 6.611 ± 2.619 |
| RForestProxNormAvg | 5.305 | 0.017 | 0.511 | 41191.135 | 1.255 | 2.677 | 177.182 | 143.124 | 7.394 | 6.833 ± 2.784 |
| RForestProxNormMin | 5.317 | 0.018 | 0.51 | 41058.845 | 1.255 | 2.677 | 177.182 | 143.116 | 7.416 | 7.167 ± 3.800 |
| RForestProxRankAvg | 5.279 | 0.018 | 0.51 | 41347.346 | 1.255 | 2.678 | 178.004 | 144.617 | 7.308 | 8.944 ± 2.721 |
| RForestProxRankMin | 5.307 | 0.017 | 0.509 | 41366.16 | 1.257 | 2.678 | 177.978 | 145.404 | 7.366 | 9.333 ± 3.976 |
| Random | 5.273 | 0.018 | 0.513 | 41865.799 | 1.255 | 2.679 | 178.017 | 144.487 | 7.425 | 11.167 ± 2.969 |
| OracleNormAvg | 5.329 | 0.019 | 0.516 | 41513.983 | 1.251 | 2.677 | 177.716 | 144.101 | 7.173 | 9.5 ± 3.961 |
| OracleNormMin | 5.286 | 0.018 | 0.515 | 41600.439 | 1.251 | 2.675 | 177.758 | 144.14 | 7.217 | 8.333 ± 3.865 |
| OracleRankAvg | 5.235 | 0.017 | 0.512 | 41237.003 | 1.248 | 2.677 | 177.182 | 143.032 | 6.74 | 3.722 ± 2.917 |
| OracleRankMin | 5.248 | 0.017 | 0.509 | 41078.147 | 1.25 | 2.676 | 177.182 | 142.967 | 7.013 | 2.833 ± 1.199 |
| CLUS-RF | 5.431 | 0.019 | 0.524 | 41384.376 | 1.262 | 2.678 | 177.182 | 143.124 | 7.422 | 4.889 ± 3.209 |
| MSVR | 6.337 | 0.816 | 0.168 | 45561.61 | 1.304 | 2.988 | 230.311 | 229.473 | 23.985 | 9 ± 3.000 |
| T_{OOB} | 5.414 | 0.019 | 0.518 | 41536.77 | 1.257 | 2.678 | 184.59 | 152.449 | 7.609 | 5.611 ± 1.799 |
| $T_{OOB} + \text{Airbag}$ | 5.423 | 0.018 | 0.52 | 41567.099 | 1.257 | 2.676 | 184.062 | 148.509 | 7.571 | 5.278 ± 1.679 |
| $T_{OOBInitial}$ | 5.418 | 0.017 | 0.517 | 41509.666 | 1.257 | 2.781 | 184.642 | 155.167 | 7.322 | 4.833 ± 2.634 |
| $T_{OOBInitial} + \text{Airbag}$ | 5.418 | 0.017 | 0.52 | 41585.324 | 1.257 | 2.676 | 182.042 | 145.905 | 7.468 | 4.389 ± 1.635 |
| $T_{top10\%}$ | 5.35 | 0.017 | 0.514 | 42109.113 | 1.256 | 2.753 | 185.862 | 153.104 | 7.429 | 4.722 ± 3.270 |
| $T_{top10\%} + \text{Airbag}$ | 5.451 | 0.017 | 0.524 | 41078.715 | 1.263 | 2.676 | 178.31 | 143.608 | 7.502 | 4.944 ± 3.330 |
| $T_{avg10\%}$ | 5.427 | 0.019 | 0.517 | 41721.025 | 1.257 | 2.742 | 189.001 | 161.129 | 7.51 | 6.889 ± 1.949 |
| $T_{avg10\%} + \text{Airbag}$ | 5.376 | 0.019 | 0.521 | 41386.025 | 1.263 | 2.678 | 177.726 | 143.052 | 7.424 | 4.444 ± 2.888 |

Table A.11

RMSE on the 9 considered datasets obtained with 10% of labeled data by supervised methods (CLUS-RF and MSVR), self-training with the proposed reliability scores (VarianceNormAvg, VarianceNormMin, VarianceRankAvg, VarianceRankMin, RForestProxNormAvg, RForestProxNormMin, RForestProxRankAvg and RForestProxRankMin), and self-training with benchmark scores (Random, OracleNormAvg, OracleNormMin, OracleRankAvg and OracleRankMin). The lower sub-table presents the performance of self-training with the VarianceNormAvg reliability score and several procedures for automatic selection of a threshold for the reliability score (T_{OOB} , $T_{OOBInitial}$, $Top10\%$ and $Tavg10\%$), with or without the Airbag stopping criteria. The best result for each column of each sub-table is marked in bold. The last column presents the average rank of the methods according to their performance on the 9 datasets (calculated separately for each sub-table).

| Method | Dataset | | | | | | | | | Average rank |
|---------------------------|--------------|--------------|--------------|------------------|---------------|------------------|----------------|----------------|--------------|----------------------|
| | Sigma Real | Sigma Sim. | Soil Quality | Solar Flare-2 | Water Quality | Vegetation Cond. | SCM20D | SCM1D | RF1 | |
| CLUS-RF | 4.922 | 0.015 | 0.527 | 38615.185 | 1.231 | 2.601 | 161.061 | 127.894 | 5.089 | 10.444 ± 3.948 |
| MSVR | 5.815 | 0.094 | 0.169 | 43551.48 | 1.276 | 2.998 | 220.416 | 219.555 | 22.315 | 13.444 ± 4.667 |
| VarianceNormAvg | 4.918 | 0.014 | 0.519 | 38576.336 | 1.225 | 2.602 | 161.021 | 127.894 | 4.905 | 7.444 ± 3.820 |
| VarianceNormMin | 4.913 | 0.014 | 0.519 | 38576.336 | 1.225 | 2.602 | 161.061 | 127.894 | 5.038 | 8.278 ± 3.203 |
| VarianceRankAvg | 4.901 | 0.014 | 0.517 | 38653.538 | 1.224 | 2.603 | 160.726 | 129.139 | 4.904 | 7.722 ± 3.684 |
| VarianceRankMin | 4.873 | 0.014 | 0.516 | 38551.846 | 1.225 | 2.601 | 161.312 | 128.189 | 4.955 | 6.556 ± 2.877 |
| RForestProxNormAvg | 4.904 | 0.014 | 0.518 | 38621.215 | 1.226 | 2.601 | 161.061 | 127.894 | 5.129 | 8.833 ± 2.773 |
| RForestProxNormMin | 4.904 | 0.014 | 0.519 | 38607.197 | 1.226 | 2.601 | 161.061 | 127.879 | 5.107 | 8.278 ± 3.280 |
| RForestProxRankAvg | 4.882 | 0.014 | 0.516 | 38655.623 | 1.225 | 2.601 | 162.15 | 129.465 | 4.972 | 9 ± 3.354 |
| RForestProxRankMin | 4.894 | 0.014 | 0.517 | 38602.015 | 1.227 | 2.601 | 161.273 | 128.512 | 5.036 | 8.278 ± 2.224 |
| Random | 4.861 | 0.014 | 0.519 | 39485.778 | 1.225 | 2.604 | 161.948 | 129.231 | 5.115 | 10.667 ± 4.00 |
| OracleNormAvg | 4.867 | 0.014 | 0.52 | 39189.14 | 1.223 | 2.601 | 161.687 | 128.902 | 4.929 | 8.611 ± 3.855 |
| OracleNormMin | 4.866 | 0.014 | 0.519 | 39004.731 | 1.223 | 2.6 | 161.276 | 128.937 | 4.901 | 7.111 ± 4.053 |
| OracleRankAvg | 4.877 | 0.013 | 0.516 | 38615.185 | 1.219 | 2.6 | 160.148 | 127.846 | 4.545 | 2.833 ± 2.462 |
| OracleRankMin | 4.848 | 0.013 | 0.515 | 38579.84 | 1.221 | 2.597 | 161.023 | 127.894 | 4.684 | 2.5 ± 1.458 |
| CLUS-RF | 4.922 | 0.015 | 0.527 | 38615.185 | 1.231 | 2.601 | 161.061 | 127.894 | 5.089 | 4.778 ± 2.938 |
| MSVR | 5.815 | 0.094 | 0.169 | 43551.48 | 1.276 | 2.998 | 220.416 | 219.555 | 22.315 | 9 ± 3.000 |
| T_{OOB} | 4.901 | 0.014 | 0.525 | 39178.759 | 1.225 | 2.602 | 166.109 | 130.989 | 5.264 | 4.833 ± 2.449 |
| $T_{OOB} + Airbag$ | 4.914 | 0.014 | 0.525 | 39148.626 | 1.225 | 2.603 | 164.526 | 129.572 | 5.191 | 4.833 ± 1.581 |
| $T_{OOBInitial}$ | 4.915 | 0.014 | 0.525 | 39381.705 | 1.224 | 2.605 | 167.385 | 136.241 | 5.073 | 5.167 ± 2.716 |
| $T_{OOBInitial} + Airbag$ | 4.916 | 0.014 | 0.523 | 38966.383 | 1.225 | 2.601 | 163.248 | 128.378 | 5.112 | 3.889 ± 1.341 |
| $Ttop10\%$ | 4.905 | 0.014 | 0.52 | 39856.55 | 1.228 | 2.646 | 166.137 | 134.911 | 5.02 | 5.056 ± 3.087 |
| $Ttop10\% + Airbag$ | 4.953 | 0.014 | 0.526 | 38539.778 | 1.233 | 2.603 | 161.236 | 128.107 | 5.098 | 4.889 ± 2.559 |
| $Tavg10\%$ | 4.992 | 0.015 | 0.527 | 38890.317 | 1.229 | 2.609 | 168.485 | 136.906 | 5.143 | 7.722 ± 1.787 |
| $Tavg10\% + Airbag$ | 4.954 | 0.015 | 0.526 | 38581.027 | 1.232 | 2.603 | 161.014 | 127.805 | 5.085 | 4.833 ± 3.122 |

Table A.12

RMSE on the 9 considered datasets obtained with 15% of labeled data by supervised methods (CLUS-RF and MSVR), self-training with the proposed reliability scores (VarianceNormAvg, VarianceNormMin, VarianceRankAvg, VarianceRankMin, RForestProxNormAvg, RForestProxNormMin, RForestProxRankAvg and RForestProxRankMin), and self-training with benchmark scores (Random, OracleNormAvg, OracleNormMin, OracleRankAvg and OracleRankMin). The lower sub-table presents the performance of self-training with the VarianceNormAvg reliability score and several procedures for automatic selection of a threshold for the reliability score (T_{OOB} , $T_{OOBInitial}$, $Top10\%$ and $Tavg10\%$), with or without the Airbag stopping criteria. The best result for each column of each sub-table is marked in bold. The last column presents the average rank of the methods according to their performance on the 9 datasets (calculated separately for each sub-table).

| Method | Dataset | | | | | | | | | Average rank |
|---------------------------|--------------|--------------|--------------|------------------|---------------|------------------|----------------|----------------|--------------|----------------------|
| | Sigma Real | Sigma Sim. | Soil Quality | Solar Flare-2 | Water Quality | Vegetation Cond. | SCM20D | SCM1D | RF1 | |
| CLUS-RF | 4.194 | 0.012 | 0.513 | 37562.992 | 1.214 | 2.568 | 149.976 | 120.878 | 4.014 | 10.5 ± 3.182 |
| MSVR | 5.194 | 0.219 | 0.165 | 97039.2 | 1.276 | 3.027 | 225.622 | 237.821 | 20.391 | 13.444 ± 4.667 |
| VarianceNormAvg | 3.978 | 0.012 | 0.506 | 37568.513 | 1.208 | 2.568 | 149.825 | 120.892 | 3.882 | 7.389 ± 2.837 |
| VarianceNormMin | 4.023 | 0.012 | 0.505 | 37562.992 | 1.21 | 2.567 | 149.83 | 120.713 | 3.941 | 7 ± 2.905 |
| VarianceRankAvg | 4.006 | 0.012 | 0.504 | 37509.874 | 1.209 | 2.567 | 149.763 | 121.37 | 3.805 | 5.778 ± 2.451 |
| VarianceRankMin | 4.028 | 0.012 | 0.504 | 37329.419 | 1.212 | 2.568 | 149.764 | 120.863 | 3.869 | 7.167 ± 3.152 |
| RForestProxNormAvg | 4.006 | 0.012 | 0.506 | 37326.996 | 1.209 | 2.568 | 150.034 | 120.872 | 4.049 | 7.833 ± 3.072 |
| RForestProxNormMin | 4.013 | 0.012 | 0.506 | 37202.539 | 1.21 | 2.568 | 149.976 | 120.878 | 4.038 | 8.056 ± 3.225 |
| RForestProxRankAvg | 4.011 | 0.012 | 0.504 | 37316.548 | 1.209 | 2.568 | 151.037 | 122.386 | 3.973 | 8.389 ± 3.781 |
| RForestProxRankMin | 4.013 | 0.012 | 0.505 | 37445.731 | 1.212 | 2.568 | 150.276 | 121.2 | 3.973 | 8.944 ± 1.976 |
| Random | 4.032 | 0.012 | 0.505 | 38273.226 | 1.21 | 2.572 | 150.947 | 122.107 | 4.054 | 11.722 ± 2.587 |
| OracleNormAvg | 4.035 | 0.012 | 0.507 | 37963.36 | 1.207 | 2.568 | 150.691 | 121.76 | 3.849 | 9.556 ± 4.065 |
| OracleNormMin | 4.027 | 0.012 | 0.507 | 37829.565 | 1.208 | 2.567 | 150.169 | 121.801 | 3.867 | 8.722 ± 3.589 |
| OracleRankAvg | 3.931 | 0.012 | 0.503 | 37475.782 | 1.204 | 2.558 | 147.849 | 120.768 | 3.505 | 2.944 ± 2.579 |
| OracleRankMin | 3.974 | 0.012 | 0.502 | 37309.152 | 1.207 | 2.556 | 149.657 | 120.742 | 3.666 | 2.556 ± 1.895 |
| CLUS-RF | 4.194 | 0.012 | 0.513 | 37562.992 | 1.214 | 2.568 | 149.976 | 120.878 | 4.014 | 4.5 ± 2.550 |
| MSVR | 5.194 | 0.219 | 0.165 | 97039.2 | 1.276 | 3.027 | 225.622 | 237.821 | 20.391 | 9 ± 3.000 |
| T_{OOB} | 4.144 | 0.012 | 0.513 | 37789.77 | 1.209 | 2.568 | 152.247 | 122.217 | 4.126 | 5.556 ± 1.895 |
| $T_{OOB} + Airbag$ | 4.148 | 0.012 | 0.513 | 37808.887 | 1.209 | 2.569 | 151.269 | 121.72 | 4.115 | 5.833 ± 1.369 |
| $T_{OOBInitial}$ | 4.124 | 0.012 | 0.513 | 38108.565 | 1.209 | 2.568 | 154.843 | 126.583 | 4.045 | 5.833 ± 2.332 |
| $T_{OOBInitial} + Airbag$ | 4.114 | 0.012 | 0.513 | 37653.957 | 1.209 | 2.569 | 150.23 | 120.983 | 4.022 | 4.5 ± 1.458 |
| $Ttop10\%$ | 4.076 | 0.012 | 0.507 | 38515.324 | 1.209 | 2.595 | 153.988 | 125.946 | 4.016 | 5.278 ± 2.906 |
| $Ttop10\% + Airbag$ | 4.166 | 0.012 | 0.513 | 37584.892 | 1.214 | 2.568 | 149.808 | 120.805 | 4.003 | 4.167 ± 2.622 |
| $Tavg10\%$ | 4.124 | 0.012 | 0.511 | 37734.932 | 1.209 | 2.57 | 154.939 | 126.063 | 4.053 | 5.778 ± 2.265 |
| $Tavg10\% + Airbag$ | 4.202 | 0.012 | 0.513 | 37480.141 | 1.214 | 2.569 | 150.143 | 120.799 | 3.998 | 4.556 ± 3.167 |

Table A.13

RMSE on the 9 considered datasets obtained with 20% of labeled data by supervised methods (CLUS-RF and MSVR), self-training with the proposed reliability scores (VarianceNormAvg, VarianceNormMin, VarianceRankAvg, VarianceRankMin, RForestProxNormAvg, RForestProxNormMin, RForestProxRankAvg and RForestProxRankMin), and self-training with benchmark scores (Random, OracleNormAvg, OracleNormMin, OracleRankAvg and OracleRankMin). The lower sub-table presents the performance of self-training with the VarianceNormAvg reliability score and several procedures for automatic selection of a threshold for the reliability score (T_{OOB} , $T_{OOBInitial}$, $Top10\%$ and $Tavg10\%$), with or without the Airbag stopping criteria. The best result for each column of each sub-table is marked in bold. The last column presents the average rank of the methods according to their performance on the 9 datasets (calculated separately for each sub-table).

| Method | Dataset | | | | | | | | | Average rank |
|----------------------------------|--------------|--------------|--------------|------------------|---------------|------------------|----------------|----------------|--------------|----------------------|
| | Sigma Real | Sigma Sim. | Soil Quality | Solar Flare-2 | Water Quality | Vegetation Cond. | SCM20D | SCM1D | RF1 | |
| CLUS-RF | 4.14 | 0.011 | 0.512 | 36704.708 | 1.203 | 2.544 | 142.221 | 115.072 | 3.241 | 11.333 ± 2.958 |
| MSVR | 5.076 | 0.086 | 0.157 | 41336.29 | 1.275 | 3.054 | 231.853 | 213.93 | 23.705 | 13.444 ± 4.667 |
| VarianceNormAvg | 3.953 | 0.011 | 0.507 | 36704.708 | 1.199 | 2.542 | 141.417 | 115.042 | 3.157 | 6.278 ± 3.270 |
| VarianceNormMin | 3.989 | 0.011 | 0.507 | 36516.581 | 1.2 | 2.543 | 141.502 | 115.042 | 3.187 | 7.444 ± 2.920 |
| VarianceRankAvg | 3.983 | 0.011 | 0.506 | 36442.51 | 1.2 | 2.543 | 141.661 | 115.307 | 3.08 | 7.167 ± 2.610 |
| VarianceRankMin | 3.966 | 0.011 | 0.506 | 36656.879 | 1.202 | 2.544 | 141.424 | 115.125 | 3.158 | 8.444 ± 2.242 |
| RForestProxNormAvg | 3.963 | 0.011 | 0.506 | 36477.326 | 1.2 | 2.544 | 141.592 | 115.046 | 3.283 | 7.722 ± 2.852 |
| RForestProxNormMin | 3.98 | 0.011 | 0.506 | 36479.676 | 1.201 | 2.544 | 141.811 | 115.072 | 3.279 | 8.778 ± 2.224 |
| RForestProxRankAvg | 3.944 | 0.011 | 0.504 | 36546.452 | 1.2 | 2.544 | 141.901 | 116.016 | 3.234 | 8 ± 3.363 |
| RForestProxRankMin | 3.949 | 0.011 | 0.505 | 36589.821 | 1.202 | 2.544 | 142.145 | 115.119 | 3.222 | 8.444 ± 2.811 |
| Random | 3.958 | 0.011 | 0.505 | 37550.074 | 1.2 | 2.547 | 143.082 | 116.181 | 3.305 | 10.778 ± 3.890 |
| OracleNormAvg | 4 | 0.011 | 0.507 | 37340.94 | 1.198 | 2.542 | 142.098 | 115.869 | 3.223 | 9.278 ± 3.589 |
| OracleNormMin | 4.006 | 0.011 | 0.507 | 37032.297 | 1.198 | 2.541 | 141.37 | 115.876 | 3.151 | 7.778 ± 4.583 |
| OracleRankAvg | 3.94 | 0.011 | 0.504 | 36376.513 | 1.194 | 2.528 | 138.857 | 114.778 | 2.932 | 2.167 ± 2.121 |
| OracleRankMin | 3.935 | 0.011 | 0.504 | 36440.297 | 1.198 | 2.526 | 140.932 | 115.059 | 3.001 | 2.944 ± 2.098 |
| CLUS-RF | 4.14 | 0.011 | 0.512 | 36704.708 | 1.203 | 2.544 | 142.221 | 115.072 | 3.241 | 5.333 ± 2.411 |
| MSVR | 5.076 | 0.086 | 0.157 | 41336.29 | 1.275 | 3.054 | 231.853 | 213.93 | 23.705 | 9 ± 3.000 |
| T_{OOB} | 4.079 | 0.011 | 0.51 | 36946.38 | 1.2 | 2.545 | 142.477 | 115.683 | 3.29 | 5.389 ± 1.516 |
| $T_{OOB} + \text{Airbag}$ | 4.096 | 0.011 | 0.511 | 36815.638 | 1.2 | 2.545 | 142.024 | 115.454 | 3.293 | 5.444 ± 1.488 |
| $T_{OOBInitial}$ | 4.075 | 0.011 | 0.509 | 36991.573 | 1.2 | 2.544 | 145.063 | 118.584 | 3.319 | 5.778 ± 2.740 |
| $T_{OOBInitial} + \text{Airbag}$ | 4.107 | 0.011 | 0.51 | 36811.305 | 1.2 | 2.543 | 141.772 | 115.116 | 3.231 | 3.5 ± 1.601 |
| $Ttop10\%$ | 4.004 | 0.011 | 0.507 | 37708.519 | 1.2 | 2.564 | 144.544 | 118.808 | 3.301 | 5.944 ± 3.167 |
| $Ttop10\% + \text{Airbag}$ | 4.16 | 0.011 | 0.512 | 36596.684 | 1.204 | 2.544 | 141.842 | 115.115 | 3.265 | 5.278 ± 2.980 |
| $Tavg10\%$ | 4.114 | 0.011 | 0.511 | 36582.057 | 1.2 | 2.544 | 144.564 | 118.045 | 3.269 | 5.222 ± 2.123 |
| $Tavg10\% + \text{Airbag}$ | 4.123 | 0.011 | 0.511 | 36677.801 | 1.204 | 2.543 | 141.918 | 115.01 | 3.229 | 4.111 ± 2.859 |

Table A.14

RMSE on the 9 considered datasets obtained with 30% of labeled data by supervised methods (CLUS-RF and MSVR), self-training with the proposed reliability scores (VarianceNormAvg, VarianceNormMin, VarianceRankAvg, VarianceRankMin, RForestProxNormAvg, RForestProxNormMin, RForestProxRankAvg and RForestProxRankMin), and self-training with benchmark scores (Random, OracleNormAvg, OracleNormMin, OracleRankAvg and OracleRankMin). The lower sub-table presents the performance of self-training with the VarianceNormAvg reliability score and several procedures for automatic selection of a threshold for the reliability score (T_{OOB} , $T_{OOBInitial}$, $Top10\%$ and $Tavg10\%$), with or without the Airbag stopping criteria. The best result for each column of each sub-table is marked in bold. The last column presents the average rank of the methods according to their performance on the 9 datasets (calculated separately for each sub-table).

| Method | Dataset | | | | | | | | | Average rank |
|----------------------------------|--------------|--------------|--------------|------------------|---------------|------------------|----------------|----------------|--------------|----------------------|
| | Sigma Real | Sigma Sim. | Soil Quality | Solar Flare-2 | Water Quality | Vegetation Cond. | SCM20D | SCM1D | RF1 | |
| CLUS-RF | 3.635 | 0.01 | 0.519 | 34586.338 | 1.192 | 2.507 | 130.013 | 106.865 | 2.45 | 10.611 ± 3.190 |
| MSVR | 4.796 | 0.081 | 0.151 | 38370.49 | 1.276 | 3.341 | 222.24 | 218.396 | 18.38 | 13.444 ± 4.667 |
| VarianceNormAvg | 3.613 | 0.009 | 0.513 | 34598.023 | 1.189 | 2.508 | 129.386 | 106.801 | 2.405 | 7.667 ± 3.640 |
| VarianceNormMin | 3.628 | 0.01 | 0.512 | 34586.338 | 1.19 | 2.507 | 129.221 | 106.83 | 2.439 | 8 ± 1.871 |
| VarianceRankAvg | 3.59 | 0.01 | 0.512 | 34581.316 | 1.189 | 2.509 | 128.981 | 106.769 | 2.326 | 5.944 ± 3.321 |
| VarianceRankMin | 3.627 | 0.01 | 0.512 | 34554.094 | 1.191 | 2.506 | 129.209 | 106.882 | 2.402 | 7.556 ± 3.056 |
| RForestProxNormAvg | 3.547 | 0.01 | 0.513 | 34591.64 | 1.189 | 2.507 | 129.84 | 106.783 | 2.505 | 8.278 ± 3.866 |
| RForestProxNormMin | 3.607 | 0.01 | 0.513 | 34549.428 | 1.19 | 2.507 | 129.732 | 106.766 | 2.497 | 8.444 ± 3.137 |
| RForestProxRankAvg | 3.588 | 0.01 | 0.511 | 34650.811 | 1.19 | 2.507 | 129.781 | 107.159 | 2.484 | 8.444 ± 3.292 |
| RForestProxRankMin | 3.635 | 0.01 | 0.512 | 34495.345 | 1.191 | 2.507 | 129.715 | 106.844 | 2.466 | 8.778 ± 3.289 |
| Random | 3.602 | 0.01 | 0.512 | 35459.71 | 1.19 | 2.511 | 130.77 | 107.674 | 2.526 | 11.444 ± 3.077 |
| OracleNormAvg | 3.594 | 0.01 | 0.514 | 35135.863 | 1.187 | 2.503 | 129.588 | 107.462 | 2.461 | 8.611 ± 3.903 |
| OracleNormMin | 3.592 | 0.01 | 0.512 | 34894.938 | 1.188 | 2.502 | 129.064 | 107.497 | 2.454 | 7.278 ± 3.492 |
| OracleRankAvg | 3.591 | 0.01 | 0.511 | 34504.833 | 1.185 | 2.482 | 125.819 | 105.148 | 2.246 | 2.667 ± 2.449 |
| OracleRankMin | 3.548 | 0.01 | 0.511 | 34479.713 | 1.187 | 2.483 | 127.522 | 106.642 | 2.328 | 2.833 ± 2.031 |
| CLUS-RF | 3.635 | 0.01 | 0.519 | 34586.338 | 1.192 | 2.507 | 130.013 | 106.865 | 2.45 | 4.944 ± 2.324 |
| MSVR | 4.796 | 0.081 | 0.151 | 38370.49 | 1.276 | 3.341 | 222.24 | 218.396 | 18.38 | 9 ± 3.000 |
| T_{OOB} | 3.662 | 0.01 | 0.515 | 34789.872 | 1.191 | 2.508 | 129.29 | 107.021 | 2.526 | 4.944 ± 1.810 |
| $T_{OOB} + \text{Airbag}$ | 3.627 | 0.01 | 0.519 | 34775.811 | 1.191 | 2.507 | 129.227 | 107.058 | 2.469 | 4.278 ± 2.306 |
| $T_{OOBInitial}$ | 3.684 | 0.01 | 0.518 | 34994.06 | 1.19 | 2.508 | 130.03 | 108.118 | 2.506 | 6.333 ± 2.236 |
| $T_{OOBInitial} + \text{Airbag}$ | 3.594 | 0.01 | 0.518 | 34698.755 | 1.19 | 2.508 | 129.701 | 106.747 | 2.448 | 3.222 ± 1.716 |
| $Ttop10\%$ | 3.634 | 0.01 | 0.515 | 35716.041 | 1.19 | 2.519 | 130.866 | 108.944 | 2.498 | 6.111 ± 2.987 |
| $Ttop10\% + \text{Airbag}$ | 3.679 | 0.01 | 0.519 | 34552.385 | 1.192 | 2.508 | 129.755 | 106.847 | 2.452 | 5.111 ± 2.571 |
| $Tavg10\%$ | 3.634 | 0.01 | 0.52 | 34814.006 | 1.191 | 2.509 | 129.646 | 107.423 | 2.541 | 6.389 ± 2.421 |
| $Tavg10\% + \text{Airbag}$ | 3.669 | 0.01 | 0.518 | 34556.359 | 1.192 | 2.508 | 129.902 | 106.85 | 2.437 | 4.667 ± 2.291 |

Table A.15

RMSE on the 9 considered datasets obtained with 50% of labeled data by supervised methods (CLUS-RF and MSVR), self-training with the proposed reliability scores (VarianceNormAvg, VarianceNormMin, VarianceRankAvg, VarianceRankMin, RForestProxNormAvg, RForestProxNormMin, RForestProxRankAvg and RForestProxRankMin), and self-training with benchmark scores (Random, OracleNormAvg, OracleNormMin, OracleRankAvg and OracleRankMin). The lower sub-table presents the performance of self-training with the VarianceNormAvg reliability score and several procedures for automatic selection of a threshold for the reliability score ($T_{_OOB}$, $T_{_OOBInitial}$, $Top10\%$ and $Tavg10\%$), with or without the Airbag stopping criteria. The best result for each column of each sub-table is marked in bold. The last column presents the average rank of the methods according to their performance on the 9 datasets (calculated separately for each sub-table).

| Method | Dataset | | | | | | | | | Average rank |
|------------------------------------|--------------|--------------|--------------|------------------|---------------|------------------|----------------|---------------|--------------|----------------------|
| | Sigma Real | Sigma Sim. | Soil Quality | Solar Flare-2 | Water Quality | Vegetation Cond. | SCM20D | SCM1D | RF1 | |
| CLUS-RF | 3.428 | 0.008 | 0.522 | 33734.994 | 1.175 | 2.465 | 114.366 | 96.226 | 1.792 | 10.611 ± 3.110 |
| MSVR | 4.623 | 0.161 | 0.187 | 50224.72 | 1.258 | 3.242 | 219.344 | 292.572 | 16.619 | 13.444 ± 4.667 |
| VarianceNormAvg | 3.441 | 0.008 | 0.517 | 33734.994 | 1.172 | 2.464 | 112.99 | 95.937 | 1.783 | 6.444 ± 3.147 |
| VarianceNormMin | 3.399 | 0.008 | 0.517 | 33734.994 | 1.174 | 2.464 | 113.697 | 96.141 | 1.771 | 6.5 ± 2.550 |
| VarianceRankAvg | 3.414 | 0.008 | 0.517 | 33734.994 | 1.173 | 2.465 | 112.907 | 96.026 | 1.746 | 5.944 ± 2.555 |
| VarianceRankMin | 3.426 | 0.008 | 0.517 | 33734.994 | 1.174 | 2.464 | 113.365 | 96.037 | 1.752 | 6.833 ± 2.475 |
| RForestProxNormAvg | 3.429 | 0.008 | 0.518 | 33784.707 | 1.173 | 2.465 | 114.164 | 96.189 | 1.833 | 10 ± 1.854 |
| RForestProxNormMin | 3.415 | 0.008 | 0.518 | 33732.11 | 1.174 | 2.464 | 114.366 | 96.214 | 1.834 | 9.111 ± 3.638 |
| RForestProxRankAvg | 3.414 | 0.008 | 0.518 | 33787.903 | 1.173 | 2.465 | 114.153 | 96.16 | 1.807 | 8.833 ± 2.107 |
| RForestProxRankMin | 3.417 | 0.008 | 0.518 | 33844.657 | 1.174 | 2.465 | 114.199 | 96.181 | 1.79 | 9.667 ± 1.984 |
| Random | 3.41 | 0.008 | 0.518 | 34551.197 | 1.173 | 2.467 | 114.931 | 96.607 | 1.856 | 11.111 ± 4.099 |
| OracleNormAvg | 3.418 | 0.008 | 0.516 | 34187.455 | 1.171 | 2.455 | 113.596 | 96.453 | 1.825 | 7.722 ± 4.032 |
| OracleNormMin | 3.432 | 0.008 | 0.517 | 34075.344 | 1.171 | 2.453 | 113.631 | 96.277 | 1.796 | 8.167 ± 3.649 |
| OracleRankAvg | 3.411 | 0.008 | 0.517 | 33734.994 | 1.168 | 2.429 | 109.839 | 93.7 | 1.728 | 3.167 ± 2.727 |
| OracleRankMin | 3.391 | 0.008 | 0.516 | 33723.302 | 1.17 | 2.431 | 111.569 | 95.147 | 1.731 | 2.444 ± 1.960 |
| CLUS-RF | 3.428 | 0.008 | 0.522 | 33734.994 | 1.175 | 2.465 | 114.366 | 96.226 | 1.792 | 6.222 ± 2.108 |
| MSVR | 4.623 | 0.161 | 0.187 | 50224.72 | 1.258 | 3.242 | 219.344 | 292.572 | 16.619 | 9 ± 3.000 |
| $T_{_OOB}$ | 3.404 | 0.008 | 0.522 | 34021.286 | 1.174 | 2.464 | 113.655 | 96.196 | 1.801 | 5.111 ± 2.329 |
| $T_{_OOB} + \text{Airbag}$ | 3.464 | 0.008 | 0.522 | 33820.038 | 1.174 | 2.464 | 114.086 | 96.103 | 1.785 | 5.111 ± 1.710 |
| $T_{_OOBInitial}$ | 3.422 | 0.008 | 0.521 | 34237.089 | 1.172 | 2.465 | 113.262 | 96.113 | 1.811 | 4.944 ± 2.480 |
| $T_{_OOBInitial} + \text{Airbag}$ | 3.439 | 0.008 | 0.522 | 33807.261 | 1.174 | 2.463 | 114.299 | 96.008 | 1.783 | 4.5 ± 2.291 |
| $Top10\%$ | 3.504 | 0.008 | 0.519 | 34671.671 | 1.173 | 2.469 | 114.209 | 96.69 | 1.815 | 6.556 ± 3.087 |
| $Top10\% + \text{Airbag}$ | 3.406 | 0.008 | 0.521 | 33734.61 | 1.175 | 2.464 | 114.441 | 96.113 | 1.778 | 4.444 ± 2.732 |
| $Tavg10\%$ | 3.421 | 0.008 | 0.521 | 33992.132 | 1.174 | 2.465 | 113.145 | 95.938 | 1.799 | 4.167 ± 2.151 |
| $Tavg10\% + \text{Airbag}$ | 3.456 | 0.008 | 0.522 | 33743.407 | 1.175 | 2.464 | 114.226 | 96.079 | 1.773 | 4.944 ± 2.480 |

References

- [1] O. Chapelle, B. Schölkopf, A. Zien, *Semi-supervised learning*, MIT Press, Cambridge, MA, 2006.
- [2] R. Hwa, P. Resnik, A. Weinberg, C. Cabezas, O. Kolak, Bootstrapping parsers via syntactic projection across parallel texts, *Nat. Lang. Eng.* 11 (03) (2005) 311–325.
- [3] E. Biber, The challenge of collecting and using environmental monitoring data, *Ecol. Soc.* 18 (4) (2013).
- [4] D. Demšar, S. Džeroski, T. Larsen, J. Struyf, J. Axelsen, M. Pedersen, P. Krogh, Using multi-objective classification to model communities of soil microarthropods, *Ecol. Modell.* 191 (1) (2006) 131–143.
- [5] D. Stojanova, P. Panov, V. Gjorgjioski, A. Kobler, S. Džeroski, Estimating vegetation height and canopy cover from remotely sensed data with machine learning, *Ecol. Inform.* 5 (4) (2010) 256–266.
- [6] D. Kocov, S. Džeroski, M.D. White, G.R. Newell, P. Griffioen, Using single- and multi-target regression trees and ensembles to model a compound index of vegetation condition, *Ecol. Modell.* 220 (8) (2009) 1159–1168.
- [7] U. Brefeld, *Semi-supervised Structured Prediction Models*, Humboldt-Universität zu Berlin, Berlin, Germany, 2008 PhD thesis.
- [8] Y. Zhang, D.-Y. Yeung, Semi-Supervised Multi-task Regression, in: *Machine Learning and Knowledge Discovery in Databases*, volume 5782 of LNCS, 2009, pp. 617–631.
- [9] R. Navaratnam, A. Fitzgibbon, R. Cipolla, The Joint Manifold Model for Semi-supervised Multi-valued Regression, in: *Proc. of the 11th IEEE Int'l Conf. on Computer Vision*, 2007, pp. 1–8.
- [10] U. Brefeld, T. Gärtner, T. Scheffer, S. Wrobel, Efficient Co-Regularised Least Squares Regression, in: *Proc. of the 23rd Int'l Conf. on Machine learning*, 2006, pp. 137–144.
- [11] Z.-H. Zhou, M. Li, Semi-supervised regression with co-training style algorithms, *IEEE Trans. Knowl. Data Eng.* 19 (11) (2007) 1479–1493.
- [12] A. Appice, M. Ceci, D. Malerba, An Iterative Learning Algorithm for Within-network Regression in the Transductive Setting, in: *Discovery Science*, volume 5808 of LNCS, 2009, pp. 36–50.
- [13] M.-C. Yang, Y.-C.F. Wang, A self-learning approach to single image super-resolution, *IEEE Trans. Multimedia* 15 (3) (2013) 498–508.
- [14] D. Kocov, C. Vens, J. Struyf, S. Džeroski, Tree ensembles for predicting structured outputs, *Pattern Recognit.* 46 (3) (2013) 817–833.
- [15] J. Levatić, D. Kocov, S. Džeroski, The importance of the label hierarchy in hierarchical multi-label classification, *J. Intell. Inf. Syst.* 45 (2) (2014) 247–271.
- [16] X. Zhu, *Semi-Supervised Learning Literature Survey*, Technical Report, Computer Sciences, University of Wisconsin-Madison, 2008.
- [17] A. Appice, S. Džeroski, Stepwise Induction of Multi-target Model Trees, in: *Machine Learning: ECML 2007*, volume 4701 of LNCS, 2007, pp. 502–509.
- [18] J. Struyf, S. Džeroski, Constraint Based Induction of Multi-objective Regression Trees, in: *Knowledge Discovery in Inductive Databases*, volume 3933 of LNCS, 2006, pp. 222–233.
- [19] L. Breiman, Random forests, *Mach. Learn.* 45 (1) (2001) 5–32.
- [20] Z. Bosnić, I. Kononenko, Comparison of approaches for estimating reliability of individual regression predictions, *Data Knowl. Eng.* 67 (3) (2008) 504–516.
- [21] M. Sánchez-Fernández, M. de Prado-Cumplido, J. Arenas-García, F. Pérez-Cruz, Svm multiregression for nonlinear channel estimation in multiple-input multiple-output systems, *IEEE Trans. Signal Process.* 52 (8) (2004) 2298–2307.
- [22] W.J. Brouwer, J.D. Kubicki, J.O. Sofo, C.L. Giles, An investigation of machine learning methods applied to structure prediction in condensed matter, *arXiv preprint arXiv:1405.3564* (2014).
- [23] J. Levatić, M. Ceci, D. Kocov, S. Džeroski, Semi-Supervised Learning for Multi-target Regression, in: *New Frontiers in Mining Complex Patterns, ECML/PKDD Workshop*, 2014, pp. 110–123.
- [24] J. Levatić, M. Ceci, D. Kocov, S. Džeroski, Semi-Supervised Learning for Multi-target Regression, in: A. Appice, M. Ceci, C. Loglisci, G. Manco, E. Masciari, Z.W. Ras (Eds.), *New Frontiers in Mining Complex Patterns*, volume 8983 of LNCS, Springer International Publishing, 2015, pp. 3–18.
- [25] A. Blum, T. Mitchell, Combining Labeled and Unlabeled Data with Co-training, in: *Proc. of the 11th Annual Conf. on Computational Learning Theory*, 1998, pp. 92–100.
- [26] D. Yarowsky, Unsupervised Word Sense Disambiguation Rivaling Supervised Methods, in: *Proc. of the 33rd Annual Meeting of the Association for Computational Linguistics*, 1995, pp. 189–196.
- [27] C. Rosenberg, M. Hebert, H. Schneiderman, Semi-Supervised Self-training of Object Detection Models, in: *Proc. of the 7th IEEE Workshop on Applications of Computer Vision*, 2005.
- [28] C. Leistner, A. Saffari, J. Santner, H. Bischof, Semi-Supervised Random Forests, in: *Proc. of the 12th Int'l Conf. on Computer Vision*, 2009, pp. 506–513.
- [29] E. Riloff, J. Wiebe, T. Wilson, Learning Subjective Nouns Using Extraction Pattern Bootstrapping, in: *Proc. of the 7th Conf. on Natural Language Learning*, 2003, pp. 25–32.
- [30] J. Bandouch, O.C. Jenkins, M. Beetz, A self-training approach for visual tracking and recognition of complex human activity patterns, *Int. J. Comput. Vis.* 99 (2) (2012) 166–189.
- [31] S. Abney, Understanding the yarowsky algorithm, *Comput. Ling.* 30 (3) (2004) 365–395.

- [32] G. Haffari, A. Sarkar, Analysis of semi-supervised learning with the yarowsky algorithm, in: Proc. of the 23rd Conf. on Uncertainty in Artificial Intelligence, AUAI Press, pp. 159–176.
- [33] R. Sousa, J. Gama, Online Semi-supervised Learning for Multi-target Regression in Data Streams Using AMRules, Springer International Publishing, pp. 123–133.
- [34] J. Duarte, J. Gama, Multi-Target Regression from High-speed Data Streams with Adaptive Model Rules, in: Proceeding of the IEEE International Conference on Data Science and Advanced Analytics, 2015, pp. 1–10.
- [35] Y. Liu, R. Jin, L. Yang, Semi-Supervised Multi-label Learning by Constrained Non-negative Matrix Factorization, in: Proceedings of the National Conference on Artificial Intelligence, volume 21, 2006, pp. 421–426.
- [36] G. Chen, Y. Song, F. Wang, C. Zhang, Semi-Supervised Multi-label Learning by Solving a Sylvester Equation, in: Proceedings of the SIAM International Conference on Data Mining, 2008, pp. 410–419.
- [37] Y. Guo, D. Schuurmans, Semi-Supervised Multi-label Classification, in: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, 2012, pp. 355–370.
- [38] H. Borchani, G. Varando, C. Bielza, P. Larrañaga, A survey on multi-output regression, Wiley Interdiscip. Rev. 5 (5) (2015) 216–233.
- [39] P.J. Brown, J.V. Zidek, Adaptive multivariate ridge regression, Ann. Stat. 8 (1980) 64–74.
- [40] L. Breiman, J.H. Friedman, Predicting multivariate responses in multiple linear regression, J. R. Stat. Soc. 59 (1) (1997) 3–54.
- [41] S. Xu, X. An, X. Qiao, L. Zhu, L. Li, Multi-output least-squares support vector regression machines, Pattern Recognit. Lett. 34 (9) (2013) 1078–1084.
- [42] M. Pugelj, S. Džeroski, Predicting Structured Outputs K-Nearest Neighbours Method, in: Discovery Science, volume 6926 of LNCS, 2011, pp. 262–276.
- [43] T. Aho, B. Ženko, S. Džeroski, T. Elomaa, Multi-target regression with rule ensembles, J. Mach. Learn. Res. 13 (1) (2012) 2367–2407.
- [44] G. Tsoumakas, E. Spyromitros-Xioufis, A. Vrekou, I. Vlahavas, Multi-Target Regression via Random Linear Target Combinations, in: Machine Learning and Knowledge Discovery in Databases, volume 8726 of LNCS, 2014, pp. 225–240.
- [45] S. Briesemeister, J. Rahnenführer, O. Kohlbacher, No longer confidential: estimating the confidence of individual regression predictions, PLoS ONE 7 (11) (2012) e48723.
- [46] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
- [47] J.G. Carney, P. Cunningham, U. Bhagwan, Confidence and Prediction Intervals for Neural Network Ensembles, in: International Joint Conference on Neural Networks, volume 2, IEEE, 1999, pp. 1215–1218.
- [48] P. Bühlmann, Handbook of Computational Statistics, Springer-Verlag, pp. 985–1022.
- [49] L. Breiman, Out-of-bag estimation, Technical Report, University of California, 1996.
- [50] J. Tanha, M. van Someren, H. Afsarmanesh, Semi-supervised self-training for decision tree classifiers, Int. J. Mach. Learn. Cybern. (2015) 1–16.
- [51] E. Spyromitros-Xioufis, W. Groves, G. Tsoumakas, I. Vlahavas, Multi-label classification methods for multi-target regression, arXiv preprint arXiv:1211.6581(2014).
- [52] D. Demšar, M. Debeljak, C. Lavigne, S. Džeroski, Modelling Pollen Dispersal of Genetically Modified Oilseed Rape within the Field, in: Proc. of the Annual Meeting of the Ecological Society of America, 2005.
- [53] A. Asuncion, D. Newman, UCI machine learning repository, 2007.
- [54] H. Blockeel, S. Džeroski, J. Grbović, Simultaneous Prediction of Multiple Chemical Parameters of River Water Quality with Tilde, in: Principles of Data Mining and Knowledge Discovery, volume 1704 of LNCS, 1999, pp. 32–40.
- [55] M. Kuhn, K. Johnson, Applied predictive modeling, Springer, 2013.
- [56] N. Chawla, G. Karakoulas, Learning from labeled and unlabeled data: an empirical study across techniques and domains, J. Artif. Intel. Res. 23 (1) (2005) 331–366.
- [57] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
- [58] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics Bull. 1 (1945) 80–83.
- [59] P.B. Nemenyi, Distribution-free multiple comparisons, Princeton University, Princeton, NY, USA, 1963 Ph.D. thesis.
- [60] Y. Guo, X. Niu, H. Zhang, An Extensive Empirical Study on Semi-supervised Learning, in: Proc. of the 10th IEEE Conf. on Data Mining, 2010, pp. 186–195.