Journal of Machine Learning Research 1 (2012) 1-48 Submitted 9/10; Revised 12/11; Published 7/12

Multi-Target Regression with Rule Ensembles

Timo Aho[*] TIMO.AHO@IKI.FI
Department of Software Systems
Tampere University of Technology
P.O. Box 553, FI-33101 Tampere, Finland

Bernard Ženko[†] BERNARD.ZENKO@IJS.SI

Sašo Džeroski[‡†] SASO.DZEROSKI@IJS.SI
Department of Knowledge Technologies
Jožef Stefan Institute
Jamova cesta 39, SI-1000 Ljubljana, Slovenia

Tapio Elomaa TAPIO.ELOMAA@TUT.FI
Department of Software Systems
Tampere University of Technology
P.O. Box 553, FI-33101 Tampere, Finland

Editor: Carla Brodley

Abstract

Methods for learning decision rules are being successfully applied to many problem domains, in particular when understanding and interpretation of the learned model is necessary. In many real life problems, we would like to predict multiple related (nominal or numeric) target attributes simultaneously. While several methods for learning rules that predict multiple targets at once exist, they are all based on the covering algorithm, which does not work well for regression problems. A better solution for regression is the rule ensemble approach that transcribes an ensemble of decision trees into a large collection of rules. An optimization procedure is then used to select the best (and much smaller) subset of these rules and to determine their respective weights.

We introduce the FIRE algorithm for solving multi-target regression problems, which employs the rule ensembles approach. We improve the accuracy of the algorithm by adding simple linear functions to the ensemble. We also extensively evaluate the algorithm with and without linear functions. The results show that the accuracy of multi-target regression rule ensembles is high. They are more accurate than, for instance, multi-target regression trees, but not quite as accurate as multi-target random forests. The rule ensembles are significantly more concise than random forests, and it is also possible to create compact rule sets that are smaller than a single regression tree but still comparable in accuracy.

Keywords: Multi-Target Prediction, Rule Learning, Rule Ensembles, Regression

1. Introduction

In the most common machine learning setting, one predicts the value of a single target attribute, categorical or numeric. A natural generalization of this setting is to predict multiple target attributes

∗. Also in the Microtask, Tampere, Finland
†. Also in the Centre of Excellence for Integrated Approaches in Chemistry and Biology of Proteins, Ljubljana, Slovenia
‡. Also in the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia

©2012 Timo Aho, Bernard Ženko, Sašo Džeroski and Tapio Elomaa.

simultaneously. The task comes in two slightly different flavors. In *multi-target prediction* (Blockeel et al., 1998), all target attributes are (equally) important and predicted simultaneously with a single model. *Multi-task learning* (Caruana, 1997), on the other hand, originally focused on a single target attribute and used the rest for assistance only. Nowadays, however, multi-task models typically predict each target attribute individually but with at least partially distinct models.

A typical example coming from the environmental sciences is the task of predicting species distribution or community structure (Demšar et al., 2006), where we are interested in predicting the abundances of a set of different species living in the same environment. These species represent the target attributes, which might, but need not be related. Examples from other areas, ranging from natural language processing to bioinformatics and medicine are also plentiful (Jeong and Lee, 2009; Liu et al., 2010; Bickel et al., 2008).

With multiple targets, a typical solution is to create a collection of single-target models. Nevertheless, especially if we are interested in the interpretability of the model, the collection of single-target models is more complex and harder to interpret than a single model that jointly predicts all target attributes (Blockeel, 1998; Suzuki et al., 2001; Ženko and Džeroski, 2008). Furthermore, learning several tasks together may increase the predictive performance for the individual tasks due to inductive transfer, where the knowledge from one task is transfered to the other tasks (Piccart et al., 2008; Kocev et al., 2007; Suzuki et al., 2001). An additional benefit of the multi-target models is that they are less likely to overfit the data than the corresponding collections of single-target models (Blockeel, 1998; Caruana, 1997).

Rule sets, together with decision trees, are one of the most expressive and human readable model representations. They are frequently used when an interpretable model is desired. The majority of rule learning methods are based on the sequential covering algorithm (Michalski, 1969), originally designed for learning ordered rule lists for binary classification domains. This is also the case with the existing methods for learning multi-target rules (Ženko and Džeroski, 2008; Ženko, 2007). Unfortunately, on both single-target and multi-target regression problems, the accuracy of rule sets that are learned by the sequential covering approach is considerably lower than that of other regression methods, like for example, regression trees (for an empirical comparison seeŽenko, 2007).

An alternative rule learning method that performs well also on (single-target) regression problems is the approach of *rule ensembles* (Friedman and Popescu, 2005, 2008; Dembczyński et al., 2008). It creates a collection of rules and uses an optimization procedure with the purpose of finding a small (and therefore interpretable) subset of rules. Optionally, rules can be combined with simple linear functions of descriptive attributes.

In this paper, we introduce FIRE, an algorithm for multi-target regression based on the rule ensembles approach. On one hand, we compare our approach with multi-target random forests (Kocev et al., 2007), which yield good accuracy at the expense of (very) large models. On the other hand, we compare it with multi-target regression trees (Blockeel et al., 1998) and multi-target model trees (Appice and Džeroski, 2007), both of which give rise to small models of (only) moderate accuracy. Our approach provides a solution that lies in between these two extremes: It produces accurate multi-target regression models that are significantly more concise than random forests. In addition, the algorithm enables us to adjust the trade-off between the interpretability and accuracy of the learned models as desired.

An early version of the FIRE algorithm has been presented in a conference paper (Aho et al., 2009). However, the algorithm presented in this paper adds the ability to combine rules with simple linear functions, which increases its accuracy. In addition, the paper improves several details of the

optimization procedure used. The new version stops the optimization if continuing seems unlikely to be fruitful. Moreover, instead of choosing the step size for the optimization algorithm in an ad-hoc manner, the choice is now made in a more sound manner as detailed in Appendix A.

Finally, the present paper includes a significantly extended empirical evaluation: We consider a larger collection of data sets and methods with which our algorithm is compared, including the latest rule ensemble methods (Dembczyński et al., 2008) and multi-target model trees (Appice and Džeroski, 2007), which combine tree models and linear functions. In addition, we compare our approach with a recent multi-task learning algorithm (Jalali et al., 2011).

The remainder of this article is organized as follows. Section 2 presents related work on multi-target prediction, rule learning, and rule ensembles. The FIRE algorithm for learning multi-target regression rule ensembles is introduced in Section 3. Section 4 describes the experimental evaluation setting and Section 5 reports the results of the empirical evaluation. The last section concludes and gives some directions for further research.

2. Related Work

In the *multi-target prediction* task, we are given a set of training examples E of the form (x,y), where $x = (x_1, x_2, \ldots, x_K)$ is a vector of K descriptive attributes and $y = (y_1, y_2, \ldots, y_T)$ is a vector of T target attributes. Our task is to learn a model that, given a new unlabeled instance x, can predict the values of all target attributes y simultaneously. Several standard learning methods such as neural networks, decision trees, model trees, classification rules and random forests have been extended towards multi-target prediction (Caruana, 1997; Blockeel et al., 1998; Appice and Džeroski, 2007; Suzuki et al., 2001; Ženko and Džeroski, 2008; Kocev et al., 2007).

An approach related to multi-target learning is *multi-task learning* (Caruana, 1997; Argyriou et al., 2008; Chapelle et al., 2010; Jalali et al., 2011; Rakotomamonjy et al., 2011; Parameswaran and Weinberger, 2011). In multi-task learning, the aim is to solve multiple single-target learning tasks $(x,y)_{t=1}^{T}$ with different training sets E_t (and in general with different descriptive attributes) at the same time. Multi-task learning should be able to benefit from relationships between tasks, just like multi-target prediction. The result of multi-task training is a distinct trained model $f(x^t)$ for each of the T tasks.

While it is true that multi-target and multi-task learning have some common background, there are also some clear differences between them. The most obvious one is the number of trained models: a separate model for each of the tasks versus a single model trained for the entire problem. Multi-target learning aims to predict the target features and explicitly describe their relationship with the descriptive features. Moreover, it implicitly describes the relationships among the target features. The multi-task model, on the other hand, does not specifically aim to describe the relationships between the target features.

Multi-target learning implicitly captures the dependencies among the targets and represents them in the single model generated. By going through this model, we can determine the effect of the descriptive features on all the targets, and analyze the relationships, either linear or nonlinear, between targets (or groups of targets). In case the targets are related, we can obtain information about these relationships.

To place our algorithm into a broader context, we include an up-to-date multi-task linear regression algorithm as a reference in our experiments. Most of the multi-task algorithms are originally designed for classification purposes (Chapelle et al., 2010; Rakotomamonjy et al., 2011;

Parameswaran and Weinberger, 2011), but the method by Jalali et al. (2011) is readily suitable for our regression tasks. The authors try to find a compromise between selecting important weights for separate tasks and for all tasks together. That is, they are searching for both shared features and features important for each task separately. The authors do this by using both separate elementwise L1 and block L1/Lq regularization and alternate between the two during optimization. Here L1/Lq is matrix regularization, with $q > 1$ in the latter case. Because of mixing up the two "clean" regularization terms, Jalali et al. (2011) call their method "dirty", therefore we refer to their algorithm as DIRTY.

Since our method learns regression rules, it is closely related to rule learning (Flach and Lavrač, 2003). A method for learning multi-target rules has been recently developed (Ženko, 2007; Ženko and Džeroski, 2008). It employs the standard covering approach (Michalski, 1969) and can learn ordered or unordered rule sets for classification and regression domains. Its accuracy on classification domains is comparable to other classification methods, such as (multi-target) decision trees. However, on regression domains, the approach performs significantly worse than the alternatives (Ženko, 2007).

An alternative approach to rule learning is called rule ensembles (Friedman and Popescu, 2005, 2008; Dembczyński et al., 2008). Strictly speaking, any set of (unordered) rules can be called a rule ensemble, as for example, in (Indurkhya and Weiss, 2001). In this paper, however, a rule ensemble is understood to be a set of unordered rules whose predictions are combined through weighted voting, which is the approach introduced by the RULEFIT (Friedman and Popescu, 2005, 2008) and REGENDER methods (Dembczyński et al., 2008).

The RULEFIT algorithm starts by generating a set of decision trees in much the same way as ensembles are generated in methods like bagging (Breiman, 1996) and random forests (Breiman, 2001). Because such large ensembles are hard or even impossible to interpret, all the trees are transcribed into a collection of rules, and an optimization procedure is used to select a small subset of the rules and to determine their weights. As a result, we get a relatively small set of weighted rules combined in a linear fashion. In addition to rules, we can also use descriptive attributes in the linear combination if we add them to the initial set of rules, and likewise determine their weights in the optimization step. The final prediction for a given example is obtained by a weighted voting of all linear terms and those rules that apply (cover the example). The resulting model can thus be written as:

$$\hat{y} = f(x) = w_0 + \sum_{i=1}^{M} w_i r_i(x) + \underbrace{\sum_{j=1}^{K} w_{(M+j)} x_j}_{\text{optional}}, \tag{1}$$

where w_0 is the baseline prediction, the first sum is the correction value obtained from the M rules, and the second sum is the correction value obtained from the (optional) K linear terms. The rules r_i are functions, which have a value of 1 for all examples that they cover, and 0 otherwise. During the learning phase, all the weights w_i are optimized by a gradient directed optimization algorithm. The linear terms part of the model is global, that is, it covers the entire example space. Note that this is different from model trees (Quinlan, 1992; Karalič, 1992; Wang and Witten, 1997), where we may also have local linear models in tree leaves, where each such model only applies to the specific examples covered by the leaf.

Input: training examples E
Output: rules and linear terms P with their weights W

1: $D \leftarrow \text{GenerateSetOfTrees}(E)$
2: $R \leftarrow \text{ConvertTreesToRules}(D)$
3: $P \leftarrow R \cup \text{LinearTerms}(E)$ {Optional}
4: $ERR_{\min} \leftarrow \infty$
5: **for** $\tau = 1.0$ **to** 0.0 **with** *step* **do**
6: $(W_\tau, ERR_\tau) \leftarrow \text{OptimizeWeights}(P, E, \tau)$
7: **if** $ERR_\tau < ERR_{\min}$ **then**
8: $(W_{\text{opt}}, ERR_{\min}) \leftarrow (W_\tau, ERR_\tau)$
9: **else if** $ERR_\tau > threshold \cdot ERR_{\min}$ **then**
10: **break**
11: **end if**
12: **end for**
13: $(P, W) \leftarrow \text{RemoveZeroWeightedTerms}(P, W_{\text{opt}})$
14: **return** (P, W)

Figure 1: The algorithm FIRE for learning rule ensembles for multi-target regression.

3. Learning Rule Based Ensembles for Multi-Target Regression

Our algorithm for learning rule based ensembles for multi-target regression problems (which we call FIRE: Fitted rule ensembles) is greatly influenced by the RULEFIT method. The top level of the FIRE algorithm is outlined in Figure 1. It starts by generating a set of diverse regression trees. To add more diversity among the base models, the trees are converted to rules. Because linear dependencies are known to be difficult to approximate with rules, we optionally add linear terms (simple linear functions) of all numeric descriptive attributes to the collection.

FIRE then optimizes the weights of rules and linear terms with a gradient directed optimization algorithm. This optimization procedure depends on a gradient threshold parameter τ; we repeat the optimization for different values of τ in order to find a set of weights with the smallest validation error. In the end, we remove all the rules and linear terms whose weights are zero.

The resulting rule ensemble is a vector function *f;* given an unlabeled example x it predicts a vector $\hat{y}$ consisting of the values of all target attributes:

$$\hat{y} = f(x) = w_0\, avg + \sum_{i=1}^{M} w_i r_i(x) + \underbrace{\sum_{t=1}^{T}\sum_{j=1}^{K} w_{(t,j)} x_{(t,j)}}_{\text{optional}}. \qquad (2)$$

Note that this vector function is an extension of the scalar function (Equation 1) to the case of multi-target regression. The first term in Equation 2 includes a constant vector avg, whose components are the average values for each of the targets. The first sum is the contribution of the M rules: each rule r_i is a vector function that gives a constant prediction (for each of the targets), if it covers the example x, or returns a zero vector otherwise. The double sum is the contribution of optional linear terms. There is a term for each combination of a target and a numeric descriptive attribute, thus the total number of linear terms is the number of numeric descriptive attributes K times the number

of target attributes T. A linear term $x_{(t,j)}$ is a vector that corresponds to the influence of the j-th numerical descriptive attribute x_j on the t-th target attribute; its t-th component is equal to x_j, while all other components are zero:

$$x_{(t,j)} = (0,\ldots,\underset{t-1}{0},\underset{t}{x_j},\underset{t+1}{0},\ldots,0).$$

The values of all weights w are determined during the optimization phase, and our goal is to have as many weights equal to zero as possible.

Example 1 *Let the problem domain have eight descriptive attributes $x = (x_1,\ldots,x_8)$ and three target attributes $y = (y_1,y_2,y_3)$. A hypothetic rule ensemble that predicts all the target values of this domain simultaneously could be:*

$$\begin{aligned}\hat{y} = f(x) = {} & 0.95\,(16.2, 6.0, 21.1)\\ & + 0.34\,[\,\textsf{IF}\ (x_8 > 3.8)\&(x_6 > 7.2)\ \textsf{THEN}\ (15.9, 36.2, 14.4)]\\ & + 0.21\,[\,\textsf{IF}\ (x_3 \le 12.1)\ \textsf{THEN}\ (6.3, 50.0, -14.3)]\\ & + 0.80(x_2,0,0) + 0.11(0,0,x_2) + 0.17(0,x_5,0) + 0.22(0,0,x_5)\\ = {} & (15.4 + 0.80\,x_2,\ 5.7 + 0.17\,x_5,\ 20.0 + 0.11\,x_2 + 0.22\,x_5)\\ & + [\,\textsf{IF}\ (x_8 > 3.8)\&(x_6 > 7.2)\ \textsf{THEN}\ (5.4, 12.3, 4.9)]\\ & + [\,\textsf{IF}\ (x_3 \le 12.1)\ \textsf{THEN}\ (1.3, 10.5, -3.0)]\end{aligned}$$

It comprises a constant vector, two rules and four linear terms (of attributes x_2 and x_5), but can also be simplified to a sum of a vector of linear equations and two rules.

So far, we have only briefly mentioned two important aspects of our algorithm, the generation of the initial collection of trees, rules and linear terms, and the weight optimization procedure. We describe each of them in detail in the next two subsections.

3.1 Generation of Base Models

The basic decision tree learning method used within the GenerateSetOfTrees procedure of FIRE (Figure 1) is the predictive clustering tree learning method (Blockeel et al., 1998) that can learn multi-target regression trees. As a starting point, we use the implementation of this paradigm within the system CLUS (Blockeel and Struyf, 2002), which can learn multi-target regression trees (Struyf and Džeroski, 2006). A set of diverse trees is generated with the multi-target implementation of the random forest ensemble method (Kocev et al., 2007), modified to stop tree building when a given tree depth limit is reached.

It is well known that variability of constituent base models is essential for good accuracy of ensembles (Dietterich, 2000). In order to increase the tree (and, thus, rule) variability, we limit the depth of a particular tree in a randomized fashion as suggested by Friedman and Popescu (2008). The maximum depth of a tree generated in the m-th iteration (denoted as d_m) is computed as follows. Let the number of terminal nodes t_m of a tree m be a random variable $t_m = 2 + \lfloor \gamma \rfloor$, where γ is drawn from an exponential distribution

$$\Pr(\gamma) = \frac{\exp\left[-\gamma/(\bar{L}-2)\right]}{\bar{L}-2},$$

and the parameter $\bar{L}$ is the average number of terminal nodes in all trees. The depth limit for a tree m can now be computed as $d_m = \lceil \log_2(t_m) \rceil$, assuming that the root has a depth of 0. The average number of terminal nodes $\bar{L}$ of all trees is specified as a parameter to the algorithm. It should be emphasized that the parameter $\bar{L}$ only affects the *average* of all tree depth limits d_m and thus trees with larger depths can still be generated.[1]

All regression trees generated with the above procedure are transcribed into rules with the ConvertTreesToRules procedure. Each leaf of each tree is converted to a rule. The weights of these rules are later computed with gradient directed optimization, as described in Section 3.3.

However, before optimizing the rule weights, it is necessary to normalize the predictions of the rules. In order to equalize the importance of different rules and different targets we proceed in three separate steps: First, we simply zero-center all the targets. Second, we scale each rule with a factor that corresponds to the magnitude of the values predicted by the rule. This should equalize the effect of the rules on the optimization process. Third, we normalize the differing scales of target spaces away. This last step is in effect only during the optimization. The first and last steps of the process are trivial and are also repeated in most other optimization processes. The normalization process may seem overly complicated, but is necessary. In Appendix B, we describe in detail why it can not be omitted or simplified to a single scaling step. Let us now describe the three normalization steps in more detail.

In the first step, we zero-center all the rule target predictions by subtracting the average *avg* from each of the original rule predictions r'': $r' = r'' - avg$. The average *avg* contains the average values of the target attributes on the learning set.

In the second, more complex, step, we scale the predicted values r'_t of each target attribute t by dividing them with a factor χ:

$$r_t = \frac{r'_t}{\chi}. \tag{3}$$

We choose χ so that it is related to both the largest predicted value of the rule r' and to the standard deviation σ_t of a target attribute t. In detail, the normalization factor χ in Equation 3 is of the form

$$\chi = \frac{r'_m}{2\sigma_m}.$$

Here the target index $m \in \{1, \ldots, T\}$ of the maximum target value r'_m is defined by:

$$m = \arg\max_t \left| \frac{r'_t}{2\sigma_t} \right|.$$

In this way, we make all the rules have an equal maximal target prediction value.

Finally, the last step is normalization, which is in effect only during the optimization. In this step, we equalize the scaling differences between different target attribute spaces. Our intention is to use this normalization only temporarily, during optimization. Otherwise, the resulting model would not be applicable to real world data anymore. As usual, we do this simply by dividing the

1. In our preliminary experiments, the results varied only slightly with different constant depth limits. In addition, the optimal limit depended on the data set. Thus, a randomized limit seems like a natural choice. The algorithm performance was not sensitive to the distribution shape or values. Moreover, we did not use tree pruning methods, because they could reduce the diversity of the base models.

target attribute prediction values r_t by twice their standard deviations $2\sigma_t$:

$$r_t^* = \frac{r_t}{2\sigma_t} = \frac{r_t'}{2\sigma_t\chi} = \frac{r_t'' - avg_t}{2\sigma_t\chi}.$$

We again refer to Appendix B for a detailed justification of the normalization process.

3.2 Optional Linear Terms

From Equation 2 we recall that, in addition to rules, we can also add linear terms to the rule ensemble. As already mentioned, a single linear term is defined as

$$x''_{(t,j)} = (0, \ldots, \underset{t-1}{0}, \underset{t}{x''_j}, \underset{t+1}{0}, \ldots, 0).$$

Linear terms are normalized in a similar way as rules. We again shift the linear terms by the average $\overline{x''_j}$ of the j-th descriptive attribute $x'_{(t,j)} = (0, \ldots, x''_j - \overline{x''_j}, \ldots, 0)$. However, we continue by normalizing the terms to the target attribute scale

$$x_{(t,j)} = x'_{(t,j)} \frac{\sigma_t}{\sigma_j}.$$

Linear terms normalized like this appear in the final rule ensemble model.

Analogously to the third stage of rule normalization we also scale the target dimension space out temporarily:

$$x^*_{(t,j)} = \frac{x_{(t,j)}}{2\sigma_t} = \frac{x'_{(t,j)}}{2\sigma_j}.$$

This is, again, only intended to equalize the terms referring to different target attributes during the optimization procedure. See Appendix B for details.

3.3 Gradient Descent Weight Optimization

The weights from Equation 2 are determined within the OptimizeWeights procedure presented in Figure 2. The optimization problem that we address is typically formulated as:

$$\arg\min_{w} \sum_{(x,y)\in E} L\left(w_0\, avg + \sum_{i=1}^{M} w_i r_i(x) + \sum_{t=1}^{T}\sum_{j=1}^{K} w_{(t,j)} x_{(t,j)}, y\right) + \lambda \sum_{i=1}^{M} |w_i|^{\alpha}, \tag{4}$$

where L is the loss function and the last term is the regularization part. The purpose of the regularization part is to make as many weights equal to zero as possible, which means that the resulting rule ensemble will be as small as possible.

The regularization part $\sum_{i=1}^{M} |w_i|^{\alpha}$ in Equation 4 forces the weights to be smaller and adds stability to the optimization procedure. Popular values for α include $\alpha = 2$ (L2 or ridge regularization, see Vapnik, 1995) and $\alpha = 1$ (L1 or lasso regularization, see Tibshirani, 1996). The best suited value depends on the data set and the user's needs. We are interested in lasso type solutions because, as explained later, they result in models having some desired properties. Unfortunately, lasso optimization is considered to be computationally complex and thus the methods that use it have a tendency to be quite slow (Yuan et al., 2010). In our case, this is especially problematic, since the

Input: base models P, training examples E and gradient threshold parameter τ
Output: weights W and an error estimate ERR
Constant: the gradient step size β

$W_0 = \{0,0,\ldots,0\}$
$(E_t, E_v) \leftarrow \text{SplitSet}(E)$ {Training and validation}
for $i = 0$ to Maximum number of iterations **do**
  if i is multiple of 100 **then**
    $ERR_i \leftarrow \text{Error}(E_v, P, W_i)$
    if $ERR_i > threshold \cdot \min_{j<i} ERR_j$ **then**
      break
    else if $ERR_i < \min_{j<i} ERR_j$ **then**
      $\text{StoreWeights}(W_i, ERR_i)$
    end if
  end if
  $G \leftarrow \text{ComputeGradients}(E_t, P, W_i)$
  if Limit of allowed nonzero weights is reached **then**
    $G \leftarrow \{g_k \in G | w_k \in W_i : w_k \neq 0\}$
  end if
  $G_{\max} \leftarrow \{g_j \in G | \; |g_j| \geq \tau \max_k |g_k|\}$
  $W_{i+1} \leftarrow W_i - \beta G_{\max}$
end for
$(W, ERR) \leftarrow \text{WeightsWithSmallestError}(E_v, P)$
return (W, ERR)

Figure 2: The procedure OptimizeWeights for gradient directed optimization.

multi-target setting increases the size of the optimization problem (the number of variables) significantly. While lasso-like regularization in multi-task optimization has been under extensive research recently (Argyriou et al., 2008; Jalali et al., 2011; Rakotomamonjy et al., 2011), its computational complexity remains an issue of concern. For computational complexity reasons, we decided to use a simple but efficient optimization procedure, which performs implicit adaptive regularization. We aim for the same goals as explicit regularization, that is, minimizing the error and achieving a sparse solution, but do not include an explicit regularization term in the optimized function. We follow the approach of Friedman and Popescu (2004), where sparsity is achieved by allowing only a small number of weights (all originally set to zero) to change. Below we describe our optimization approach in detail.

Friedman and Popescu (2004) show, that for the gradient directed optimization, an effect very similar to the effect of explicit regularization can also be achieved in a different and more efficient way, without solving the optimization problem directly. They propose a gradient directed optimization method with squared loss

$$L_{\text{sqrd}}(f_t(x), y_t) = \frac{1}{2}(f_t(x) - y_t)^2,$$

where $f_t(x)$ is the predicted value and y_t is the true value. The square loss function is often used for solving such optimization problems, but is only applicable to single-target problems.

If we want to use a similar gradient directed optimization algorithm for multi-target problems, we have to define a suitable loss function that is convex. A typical solution is to take the above squared loss function for each of the T target attributes and aggregate the per-target losses by taking their average:

$$L(f(x),y) = \frac{1}{T}\sum_{t=1}^{T} L_{\text{sqrd}}(f_t(x),y_t). \tag{5}$$

Such an aggregated loss function is convex and allows for efficient computation of the gradients.

Another possibility is, for example, the maximum value of the single-target loss functions. However, our preliminary experiments showed that this results in larger and less accurate models. In addition, this loss function would result in a significantly slower algorithm because, in addition to the gradients, we would also have to compute the loss function values for each target explicitly.

Instead of adding a regularization term to the optimization problem, we explicitly control the number of weights that are changed during every optimization iteration in the following way. Let M be the number of weights we are optimizing with a gradient method. Instead of allowing changes to all the weights simultaneously, we only allow changes to the weights w_j whose gradients g_j have a value above a threshold

$$|g_j| \geq \tau \cdot \max_k |g_k|.$$

If $\tau = 0$, we are changing all the weights during every iteration, resulting in a behavior similar to ridge regularization ($\alpha = 2$). On the other hand, if $\tau = 1$, only one gradient during every iteration is modified and the behavior is similar to lasso regularization ($\alpha = 1$). In our case, lasso regularization seems to be the best choice, because it has been shown to lead to many weights being set to zero (Tibshirani, 1996), which means simpler and more interpretable models with fewer rules.

In practice it is hard to know in advance which value of τ will result in the most accurate model. Both theoretical and experimental results suggest that different data sets are best suited by different regularizations (Lounici et al., 2009; Rakotomamonjy et al., 2011). Thus, the most suitable value of τ depends on the properties of the learning data. We overcome this problem by trying a set of different values of τ (Figure 1, line 5) and estimating their accuracies on a separate internal validation set E_v (Figure 2, line 19), which is the same for all τ values. In the end, the model with the smallest validation error is selected.

Our aim is to efficiently learn a rule ensemble model that is both small and accurate. Therefore, we start with a τ value that creates a small model, $\tau = 1$, and then iteratively decrease the value of τ until it reaches zero. We stop the loop if the validation error stops decreasing, since it is unlikely that trying smaller values would result in a more accurate model.

It is possible that we are stopping in only a local optimum and the result can be a suboptimal model. However, in practice, not evaluating the lower τ values does not seem to lower the accuracy significantly. Also, this procedure is very effective, because most of the optimization time, namely $|E|(M+TK)^2/2$, is spent on computing the covariance matrix of weights. Here $|E|$ is the learning set size and $M+TK$ the number of optimized weights. Nevertheless, we do not have to compute the covariances for zero weighted predictive terms. Thus, most of the resources are usually used for optimization with lower values of τ. For example, in practice, the case $\tau = 0$ seems to use at least half of the computing time alone.

The complete optimization procedure OptimizeWeights is presented in Figure 2. It starts by initializing all the weights to zero and splitting the entire learning set E into a learning set E_t and an internal validation set E_v. Within the loop, we iteratively compute the gradients g_k for each of the

weights (line 12) and then change the selected weights w_j in the most promising direction $-G_{\max}$ for a predefined step size β (line 17). The step size is an automatically computed constant, which is based on the theoretically optimal step. See Appendix A for a more detailed description of the step size computation.

In addition to this basic idea, there are some additional details. First, on every 100-th iteration we stop the optimization if we are overfitting (lines 6–7), that is, if the validation error starts to increase. Second, we can define a maximum number of nonzero weights in advance (lines 13–15), which makes a suitable parameter for setting the accuracy-for-simplicity trade-off. An extensive experimental evaluation of the algorithm's performance is presented in the next two sections.

4. Experimental Setup

In the experimental evaluation, we investigate three different issues. First, we evaluate our algorithm FIRE on single-target regression domains in order to show that our algorithm is also applicable to standard regression problems and that its performance on such problems is comparable to the performance of other tree and rule based regression methods. We compare FIRE with regression trees (Breiman et al., 1984), random forests (Breiman, 2001), model trees (Quinlan, 1992; Karalič, 1992), the L1/Lq regularized multi-task regression algorithm DIRTY (Jalali et al., 2011), the rule ensemble methods RULEFIT (Friedman and Popescu, 2005, 2008) and REGENDER (Dembczyński et al., 2008). In the comparison, we focus on the accuracy and size of the learned models. The model size is used to indicate the interpretability of the model.

Second, we evaluate FIRE on multi-target domains, that is, on the problems for which it was designed in the first place. We compare it with three other multi-target variants of popular tree based methods: regression trees (Blockeel et al., 1998), random forests (Kocev et al., 2007), and model trees (Appice and Džeroski, 2007). In addition, we use the L1/Lq regularized multi-task regression algorithm DIRTY (Jalali et al., 2011) for a reference. This is the main part of the evaluation since it was designed to show how effective FIRE is when compared with other state-of-the-art multi-target prediction methods. Again, we focus on the accuracy and size of the learned models.

As described in Section 3.3, FIRE has a parameter that can be used to limit the total number of nonzero weights, that is, the number of rules and linear terms. We use this parameter in the third part of the evaluation to investigate how the size of the rule ensemble influences its accuracy. The preliminary experiments showed that, in some cases, we can significantly reduce the model size with only a marginal decrease in accuracy. This is the reason that in both above mentioned evaluations we also include results for FIRE models with an arbitrary limit of 30 rules and terms (denoted as "Max 30" in the results). Another optional setting of the FIRE (and RULEFIT) algorithm(s) is whether to include linear terms in the model or not. We present both cases in all evaluation scenarios; models with linear terms are denoted as "+ linear". In the remainder of this section, we present the algorithms and their parameter settings, evaluation methodology and data sets used in the empirical evaluation.

4.1 Algorithms and Parameters

When generating the initial set of trees (Figure 1, line 1) we use 100 random trees with an average depth of 3. The optimization procedure is run with the gradient threshold parameter τ values ranging from 1 to 0 in 0.1 decrements (Figure 1, line 5). In the OptimizeWeights procedure, the initial set E is split into $2/3$ for training E_t and $1/3$ for validation E_v (Figure 2, line 2). The maximum number

of optimization iterations is 10,000. The threshold for detecting the error increase (Figure 1, line 9 and Figure 2, line 6) is 1.1, the step size β (Figure 2, line 17) is computed automatically based on the optimal step size as described in Appendix A. Our algorithm, as well as the regression trees (Blockeel et al., 1998) and random forests (Kocev et al., 2007) used in our experiments are implemented in the CLUS predictive clustering system (Blockeel and Struyf, 2002).[2] All the parameters for regression trees and random forests are set to their default values; regression trees use reduced error pruning, random forests consist of 100 trees.

For experiments with model trees, we use the multi-target implementation MTSMOTI by Appice and Džeroski (2007) with the recommended settings: First we set the stopping criterion so that a non-leaf node covers at least a tenth of the training set. Also for continuous attributes all the distinct values are treated as candidate split thresholds for node tests. However, a value is not a candidate threshold if it causes either child to cover less than 15 examples. Finally, the maximum number of variables included in the regression model of a pruned leaf is set to 10. We report the pruned trees. The implementation of MTSMOTI does not handle missing values, so we pre-process the data by replacing them with averages or the most common labels.

Experiments with the rule ensemble methods RULEFIT and REGENDER were performed with the original implementations by the authors and the settings that lead to the best performance in the original papers (Friedman and Popescu, 2008, 2004; Dembczyński et al., 2008). For RULEFIT, we use the Huber distance with a trimming factor of 0.9. The linear variable conditioning trimming factor is set to 0.025, the average number of tree terminal nodes to 6, the maximum number of rules to 5,000, the incentive factor for using fewer variables in tree based rules to the default value 3.0, the model memory parameter value to 0.01, and the sampling fraction for the trees to $|E|/5$. The regularization parameter τ is chosen with full *internal* cross-validation. Hence, the used parameter τ value should be the best one, but may be slow to find. The maximum step size for the gradient descent algorithm is set to the default of 0.01. The convergence factor is set to 1.1. For a detailed analysis of the differences in the settings of RULEFIT and FIRE see Appendix C. The number of rules for REGENDER is set to 200, the shrinkage amount to 0.5, the data is resampled without replacement and missing values are not replaced, which is the default setting. Moreover, the minimization technique is set to gradient descent with squared error loss.

For the multi-task learning algorithm DIRTY (Jalali et al., 2011) we slightly modified the original R code by the authors. Recall that the multi-task problem domain consists of multiple single-target learning tasks in the separate training sets E_t: $\{(x,y) \in E_t | t = 1 \dots T\}$. In our multi-target data sets, however, the descriptive parts x of example sets are all the same: there is a single example $(x', (y'_t)_{t=1}^T) = (x, y') \in E$ in the multi-target setting corresponding to a collection of instances $\{(x', y'_t) \in E_t | t = 1 \dots T\}$ in the multi-task setting. In practice, we modified the code of DIRTY so that it uses the same descriptive features x for all the tasks.

Otherwise we used the parameter values and methodology suggested by Jalali et al. (2011, Appendix H) and by Ali Jalali in personal communication: For the optimization stopping criterion we set $\varepsilon = 10^{-10}$. As in RULEFIT, we used internal 10-fold cross-validation for selecting the best values for the regularization weights λ_b and λ_b. As suggested by Jalali et al. (2011, Appendix H) we used the resulting matrices B and S of previous λ combination as the initial weight matrices for the next combination. The tried λ values were of the form $c\sqrt{\log(TK)/|E|}$, where K is the number of descriptive attributes. For λ_b, the constant c has seven values $c_b \in$

2. Available at `http://clus.sourceforge.net` under the GNU General Public License.

$\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1\}$ and for λ_s similarly $c_s = c_b\sqrt{T}$ for each value of c_b. Thus, we have 49 λ value pairs to try out. Finally, the descriptive features were normalized by scaling with the maximum feature norm of the whole set: $\max_{j=1\ldots K}\sqrt{\sum_{(x,y)\in E} x_j^2}$, where $x = (x_1, x_2, \ldots, x_K)$. Thus, each descriptive feature is shrank to the interval $[-1, 1]$.

4.2 Data Sets and Evaluation Methodology

The data sets used in the experiments, together with their properties are presented in Table 1. Twenty-four single-target regression data sets are taken from the following data repositories: UCI (Asuncion and Newman, 2011), Weka (2011), StatLib (2011), Delve (2011), and Torgo (2011). Publicly available multi-target data sets, however, are scarce. In addition to a single public data set from the UCI repository, we have collected ten previously analyzed data sets with the following references: DS1 (Kampichler et al., 2000), DS2 (Karalič and Bratko, 1997), DS3 (Džeroski et al., 2006), DS4 (Stojanova, 2009), DS5 (Džeroski et al., 2002), DS6 (Demšar et al., 2006), DS7 (Demšar et al., 2005), DS8 (Džeroski et al., 2005), DS9 (Gjorgjioski et al., 2008), and DS10 (Džeroski et al., 2000).

The accuracy of the learned regression models is estimated for each target attribute by computing the relative root mean squared error (RRMSE). For a single-target model $f(x)$ and an example set E, RRMSE is computed as

$$\mathrm{RRMSE}(f, E) = \sqrt{\frac{\sum_{(x,y)\in E}(f(x) - y)^2}{\sum_{(x,y)\in E}(\bar{y} - y)^2}},$$

where $\bar{y}$ is the mean value of target attribute y over dataset E. The size of tree based models (regression trees, random forests, and MTSMOTI) is measured as the number of leaves in all the trees. The size of rule ensemble models (FIRE, RULEFIT, and REGENDER) is measured as the number of rules or the sum of the number of rules and linear terms, if linear terms are used.[3] For DIRTY, we present the number of nonzero weights (weight matrix support) as was done by Jalali et al. (2011). All the above measures are estimated using 10-fold cross-validation, where the folds for each data set are the same for all the algorithms.

To test whether any of the observed differences in accuracy and size between the algorithms are significant, we followed the methodology suggested by Demšar (2006). First, we use the Friedman test to check if there are any statistically significant differences between the compared algorithms. If the answer is positive, we additionally use the Nemenyi post-hoc test to find out what these differences are, and we present them by average ranks diagrams. These diagrams show all the compared algorithms in the order of their average ranks; the best are on the right and the worst are on the left side of the diagram. The algorithms that differ by less than a critical distance for a p-value of 0.05 are connected with a horizontal bar and are not significantly different. We perform such significance testing for both the RRMSE and model size metrics.

However, when testing the differences in RRMSE for multi-target data, we have two possibilities. On one hand, we can treat each of the target attributes as an independent measurement. The argument against this option is that target attributes within one data set are probably not independent and as a result our test will show more significant differences than there actually are. On the other

3. These measurements were chosen because they relate quite naturally to the number of linear terms. For example, with the total number of tests or conditions in trees and rules this would have been problematic. However, the results were similar with both choices.

DATA SET	# EXS	% MISS VALS	# NOM ATTS	# NUM ATTS	# TAR ATTS	# ALL ATTS	SOURCE
ABALONE	4,177	0.0	1	7	1	9	UCI
AILERONS	1,533	0.0	0	40	1	41	TORGO
AUTO-MPG	398	0.2	3	4	1	8	UCI
AUTO-PRICE	159	0.0	0	15	1	16	WEKA
BREAST CANCER	286	0.4	8	1	1	10	WEKA
CENSUS	22,784	0.0	0	8	1	9	DELVE
CLOUD	108	0.0	2	4	1	7	UCI
CPU ACTIVITY	8,192	0.0	0	12	1	13	DELVE
COMPUTER HW	209	0.0	1	6	1	8	UCI
DELTA-AILERONS	7,129	0.0	0	5	1	6	TORGO
DIABETES	43	0.0	0	2	1	3	TORGO
ECHOCARDIOGRAM	130	8.3	3	6	1	10	WEKA
HOUSING	506	0.0	1	12	1	14	UCI
HOUSING CA	20,640	0.0	0	8	1	9	STATLIB
KINEMATICS	8,192	0.0	0	8	1	9	DELVE
META-DATA	528	4.6	2	19	1	22	UCI
PBC	418	16.5	8	10	1	19	UCI
POLE TELECOMM	15,000	0.0	0	48	1	49	TORGO
PYRIMIDINES	74	0.0	0	27	1	28	TORGO
QUAKE	2,178	0.0	0	3	1	4	UCI
SENSORY	576	0.0	11	0	1	12	UCI
SERVO	167	0.0	4	0	1	5	UCI
STRIKE	625	0.0	1	5	1	7	UCI
VETERAN	137	0.0	4	3	1	8	UCI
COLLEMBOLAN	393	20.4	8	40	3	51	DS1
EDM	154	0.0	0	16	2	18	DS2
FOREST KRAS	60,607	0.0	0	160	11	171	DS3
FOREST SLIVNICA	6,219	0.0	0	149	2	151	DS4
META LEARNING	42	27.9	0	56	10	66	DS5
MICROARTHROPODS	1,944	0.1	0	142	3	145	DS6
SIGMEA REAL	817	0.0	0	4	2	6	DS7
SIGMEA SIMULATED	10,368	0.0	2	8	3	13	DS8
SOLAR FLARE	323	0.0	10	0	3	13	UCI
VEGETATION	29,679	0.0	0	64	11	75	DS9
WATER QUALITY	1,060	0.0	0	16	14	30	DS10

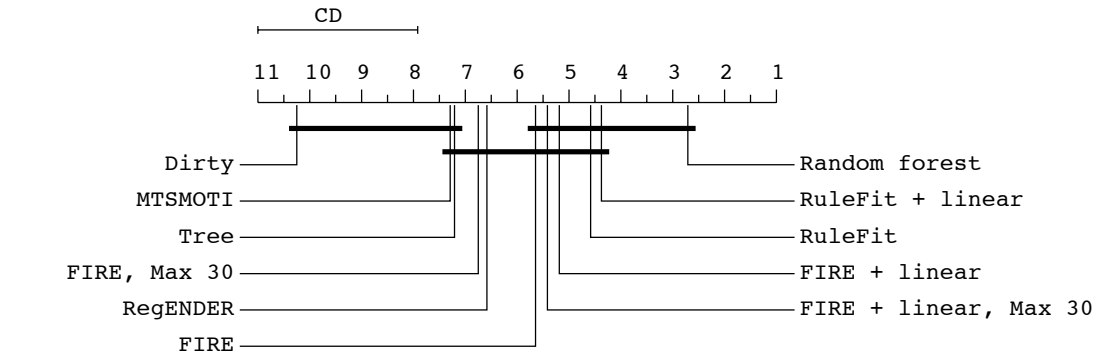
Table 1: Data sets used in the experimental evaluation and their properties. Please see the text for source references.

hand, we can compute the average over all targets within each data set and consider such averages as independent measurements. The argument against the second option is that when computing averages across all target attributes within a data set, we are actually "summing apples and oranges," and the resulting average is probably not a valid quantity. In the absence of a better solution, we present the tests of RRMSE differences for both options. The results of the experimental evaluation are presented in the next section.

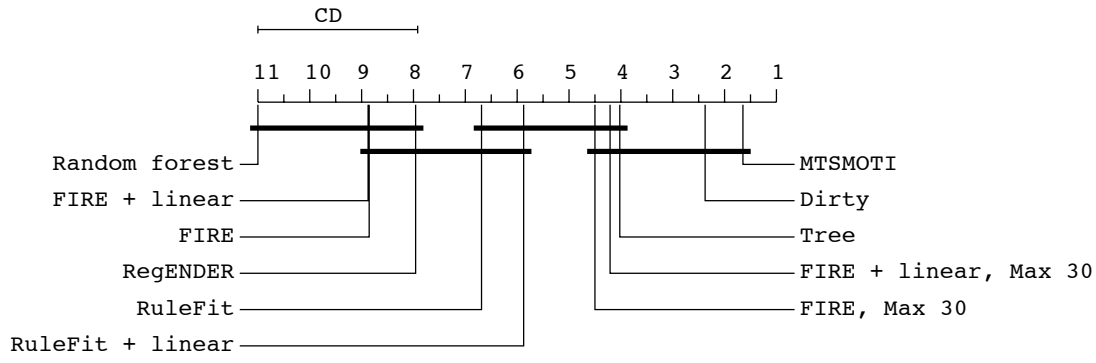
5. Results

As already mentioned, we perform three groups of experiments. We evaluate FIRE first briefly on single-target and then extensively on multi-target regression data sets. The latter is the most important part, since it shows whether our generalization of rule ensembles towards multi-target regression is successful. Next, we investigate the influence of the size of a rule ensemble on its accuracy. Finally, we present the running times for all experiments.

5.1 Single-Target Regression



(a) RRMSE



(b) model size

Figure 3: Average ranks diagrams on *single-target* data for *RRMSE* (a) and *model size* (b). Better algorithms are on the right-hand side, the ones that do not differ significantly by the Nemenyi test (p-value $= 0.05$) are connected with a horizontal bar. CD is the critical distance.

On single-target data sets, we compare regression trees, random forests, linear regression (DIRTY), model trees (MTSMOTI), and rule ensembles: four versions of FIRE, two versions of RULEFIT, and REGENDER. For FIRE and RULEFIT, we include the experiments with and without linear terms. Moreover, for FIRE we include experiments without any limitation on the model size, and experiments with the maximum number of rules and linear terms set to 30.

The Friedman test shows that the RRMSEs are statistically different with a p-value $= 6.2 \cdot 10^{-14}$ and model sizes with a p-value $< 2.2 \cdot 10^{-16}$. The average ranks for all algorithms together with the results of the Nemenyi test are given in Figure 3, separately for RRMSE and model size. The better algorithms are the ones with higher ranks (with 1 being the highest rank) and are placed on the right-hand side of the diagram. Algorithms whose ranks differ by less than a critical distance (CD) are not statistically significantly different in performance with a p-value $= 0.05$.

From the RRMSE diagram (Figure 3a), we can see that random forests are the most accurate method, followed by all the rule ensemble methods, MTSMOTI, regression trees and finally DIRTY, which seems to perform poorly on single-target data. However, there are not many statistically significant differences: Only random forests and DIRTY seem to be significantly different from part of the main group. Random forests are better and DIRTY clearly worse. The statistical similarity is somewhat surprising, considering that we have as many as 24 data sets. An intuitive reason for this is, naturally, the homogeneous background of many of the algorithms.

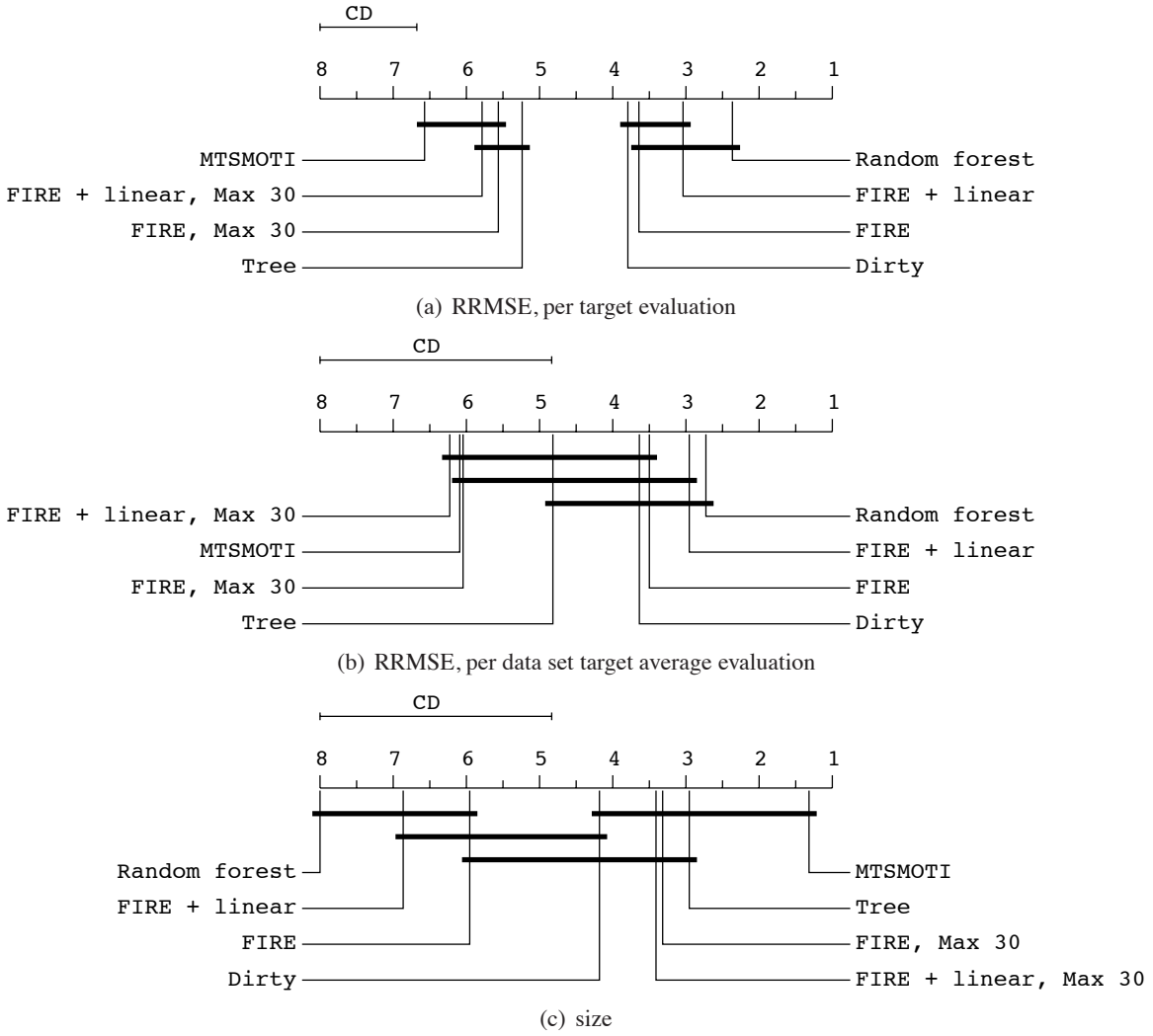
Our unlimited FIRE versions and limited FIRE with linear terms are in the middle class. However, they are slightly outperformed by RULEFIT, which is very closely related to FIRE. Not surprisingly, further experiments presented in Appendix C show that these differences are due to the RULEFIT features not implemented in FIRE. We also notice that linear terms tend to increase the accuracy of both versions of FIRE and RULEFIT. In particular, adding linear terms still yields accurate models even when we limit the number of FIRE rules. In this case, linear terms seem to be surprisingly effective, as they bring the limited FIRE of 30 terms to the same accuracy level with much larger unlimited FIRE set of rules.

The diagram for model size (Figure 3b) shows that MTSMOTI and DIRTY create the smallest models, while random forests are at the other extreme. Both unlimited FIRE versions seem to generate models that are larger than the reference rule ensembles (RULEFIT and REGENDER), but the differences are not significant. While the unlimited version of FIRE generates smaller models than random forests, the difference in size is below the significance threshold. Also the sizes of limited FIRE versions and individual regression trees seem to be very similar according to this statistical test.

The detailed results are shown in Table 4 in Appendix D. Note the large size of random forest models in comparison with all other algorithms. On average, the model size is 120,865 terminal nodes. The accuracy of random forests clearly comes with the price of increased model size. The detailed results also shed some light to the performance of limited FIRE with linear terms against unlimited FIRE without them. In a pairwise comparison, the algorithm tie in the wins over data sets (12 wins each) and the unlimited version has a slightly lower average accuracy. Thus, the result can not be explained only with the nature of Nemenyi test lacking pairwise comparison. The two algorithms really, surprisingly, seem to perform equally well while the unlimited FIRE creates much larger models.

In sum, the results of this first part of the evaluation show that our rule ensemble method FIRE performs well on single-target regression problems.

5.2 Multi-Target Regression



(a) RRMSE, per target evaluation

(b) RRMSE, per data set target average evaluation

(c) size

Figure 4: Average ranks diagrams on *multi-target* data for *RRMSE evaluated on separate targets* (a), *on target averages within data sets* (b) and *model size* (c). Better algorithms are on the right-hand side, the ones that do not differ significantly by the Nemenyi test (p-value $=$ 0.05) are connected with a horizontal bar. CD is the critical distance.

In the multi-target experiments, we use all the algorithms from the previous section that work on multi-target data sets. These are regression trees, random forests, MTSMOTI, DIRTY, and the same four versions of FIRE.

For multi-target data, the Friedman test shows that the RRMSE values of algorithms are significantly different with a p-value $< 2.2 \cdot 10^{-16}$, if we treat each target separately, and with a p-value $= 2.1 \cdot 10^{-4}$, if we compare target averages over each data set. The model sizes are different with a p-value $= 1.2 \cdot 10^{-11}$. The average ranks and results of the Nemenyi test are given in

Figure 4 for RRMSE evaluated on separate targets (a), RRMSE evaluated on target averages within data sets (b), and model size (c).

Looking at Figure 4(a), the general picture of algorithm ranking is similar to the one for single-target prediction: random forests and both unlimited FIRE versions are more accurate than DIRTY, regression trees, the limited versions of FIRE, and MTSMOTI. Due to a smaller critical distance, the four more accurate algorithms are significantly better than the rest. Evaluation over target averages within data sets (b) shows a similar picture, but because the sample size is smaller (11 data sets vs. 63 targets), the critical distance is larger and there are very few significant differences. The diagram for size (c) is very similar to the single-target case: DIRTY, the limited FIRE, regression trees, and MTSMOTI are significantly smaller than random forests.

In general, in the ranking of the algorithms according to their accuracy, the largest change from the single-target to multi-target setting is that DIRTY now performs practically as well as the unlimited FIRE without linear terms. Clearly, the moderately complex regularization of DIRTY does not pay off for single-target prediction, but does pay off for multi-target prediction. Moreover, as can be seen in the detailed results in Table 5 in Appendix D, the accuracy of DIRTY is sometimes remarkably high in comparison to the accuracies of the other algorithms. However, the performance is quite unstable and the algorithm performs poorly on several data sets. Apart from DIRTY, the results follow the single-target case in a straightforward manner: larger and more complex models are more accurate. In this sense, DIRTY is more or less an outlier—the model itself is simple but the creation process is complicated. From this point of view, its behavior being unstable but sometimes resulting in a very accurate model is intuitively understandable.

In addition, the linear terms do not seem to be as useful for the limited version of FIRE as for the single-target prediction. Moreover, the limited FIRE version performs, rather surprisingly, relatively much worse on multi-target data. Specifically, we notice this in comparison with regression trees, which were one of the least accurate models for single-target data, but they are in the middle class now. This is especially clear in per data set average evaluation and occurs in the pairwise comparison with the limited FIRE (in per target evaluation 7 out of 11 wins and per data set average 34–36 out of 63 wins for regression trees). The detailed results in Appendix D show that regression trees tend to win only when the size of the tree produced is much larger than the FIRE limit of 30. This suggests that the limit is too strict for good accuracy in general. Additional examination shows that we need a size limit of 60–80 for FIRE to overtake regression trees in terms of accuracy. The limit is, thus, much higher than the one needed in single-target case.

Surprising results are also achieved with MTSMOTI, which seems to underachieve, given the complexity of the model and the process of its induction. When we compare the results with the ones introduced in the original article (Appice and Džeroski, 2007), we notice some differences. Especially the common reference algorithm, multi-target regression tree, has radically improved results in our experiments. After studying the issue, it seems that the implementation details of the multi-target regression trees in CLUS have been modified causing the change in performance. Thus, the reason for the lower relative performance of MTSMOTI is partly due to the change of the performance of the most important reference algorithm. In our experiments, MTSMOTI model sizes are similar to those reported in the original paper. The slight changes must be due to the differences in data preprocessing and parameter values.

There are some additional interesting points in the detailed results. We note that while the difference in size between random forests and the unlimited FIRE versions is not statistically significant, the average size of a random forest (276,075 terminal nodes) is more than 300 times larger than the

average sizes of FIRE models. At the same time, the difference in average accuracy is small. When both size and accuracy are taken into account, the unlimited version of FIRE with linear terms seems to perform very well on multi-target regression problems.

We also notice that the average proportion of linear terms in the unlimited model is 24%, but drops to only 5% if we limit the model size to 30. This, and more generally the effect of linear terms on FIRE, is analyzed more in detail in Section 5.5. Nevertheless, it is also interesting that the proportion of linear terms highly depends on the data set. This suggests that for some data sets, using only rules is not enough for high accuracy. For example, FOREST SLIVNICA seem to be quite linear in nature, because MTSMOTI, and DIRTY achieve very good accuracy with moderately small models. On these data sets, the unlimited FIRE version with linear terms also seems to be better than the one without. Both MTSMOTI and DIRTY, however, seem to perform much worse on some other data sets (such as EDM).

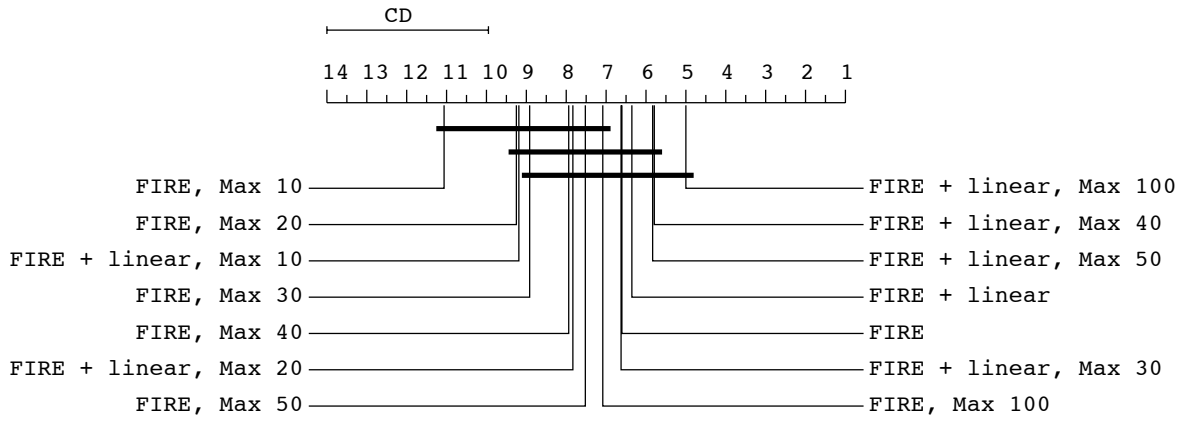
5.3 Model Size Limitation

Experiments presented in the previous two subsections considered two size-related options for the FIRE algorithm, one with the maximum model size set to 30, and one without any model size restrictions. In this subsection, we present experiments with different values of the maximum model size parameter, which show how this model size limit influences the accuracy of models. We use the values of 10, 20, 30, 40, 50, 100, and ∞.

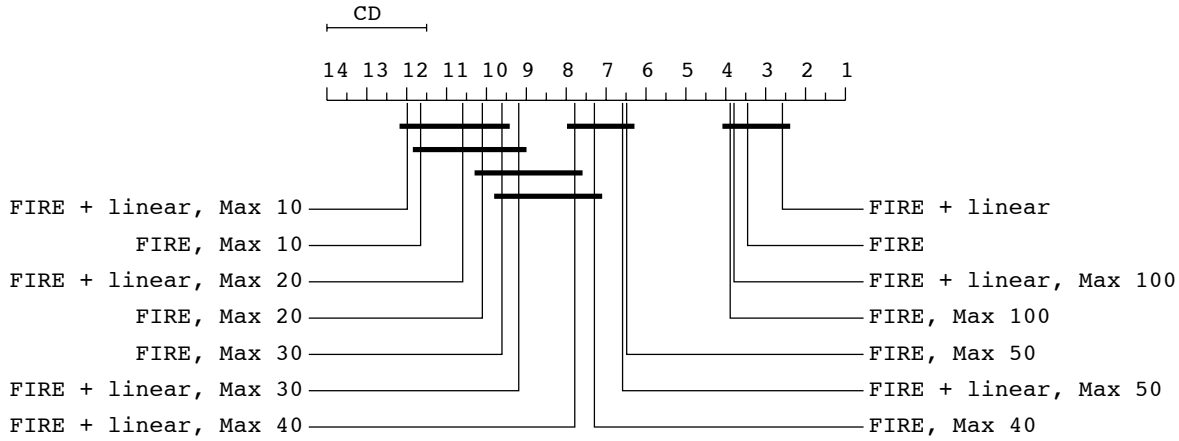
The average ranks diagrams comparing different maximum model size parameters are presented in Figure 5. Diagram (a) shows the results on single-target data. While all the differences in RRMSE are not significant, it is clear that increasing the model size improves the accuracy. Diagrams (b) and (c) show the RRMSE on multi-target data for per-target evaluation and for per-data set target average evaluation, respectively. Because of a larger sample, there are more significant differences in (b) than in (c). However, what is common to both diagrams is the trend that smaller models are also less accurate. The size limitation parameter can therefore be used as an accuracy-for-simplicity (and interpretability) trade-off setting.

Another interesting conclusion that can be drawn from these diagrams is that while a model size of 30 seems enough to get models that are not significantly less accurate than the unlimited models for single-target domains, this is not the case for multi-target domains. This clearly depends on the domain, the number of target attributes, and relations between them. The task of modeling a multi-target domain is clearly harder than the task of modeling a single-target domain, and therefore the corresponding models have to be more complex.

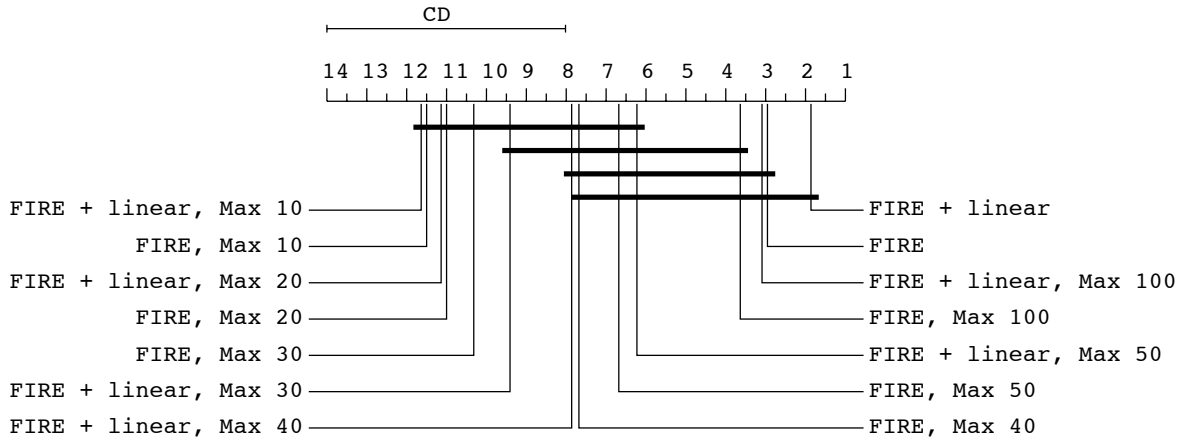
We also note that even if on single-target data linear terms always increase the accuracy of FIRE substantially, the effect is more complicated in the multi-target case. Surprisingly, for smaller size limit values (less than 30 terms and rules) the linear terms seem to decrease the accuracy. Moreover, even for larger FIRE models the effect is next to negligible. The sole exception in this is the unlimited version which clearly, although not significantly, benefits from the linear terms. We could, thus, conclude that in the multi-target case any size limit removes the gain from linear terms. On the other hand, for single-target case linear terms are always useful. We again refer to Section 5.5 for a more detailed analysis on linear terms.



(a) single-target problems



(b) multi-target problems, per target evaluation



(c) multi-target problems, per data set target average evaluation

Figure 5: The average ranks diagrams of FIRE with different model size limitations for *RRMSE* on *single-target data* (a), on *multi-target data evaluated on separate targets* (b), and on *multi-target data evaluated on target averages within data sets* (c). Better algorithms are on the right side, the ones that do not differ significantly by the Nemenyi test (p-value $=$ 0.05) are connected with a horizontal bar. CD is the critical distance.

5.4 Running Times

The running times were measured with the GNU/Linux `time` command. The underlying environment was 64 bit Red Hat Linux with 4 processors of type 2.216 GHz Dual-Core AMD Opteron, and 16 GB of memory. We ran the experiments 5 times, omitted the highest and lowest times and took the average over the three remaining measurements. The regression trees, random forests and FIRE are implemented in Java within the CLUS machine learning toolbox (Blockeel and Struyf, 2002). The REGENDER and MTSMOTI algorithms are also implemented in Java, while RULEFIT and DIRTY are implemented in the R statistical language. For RULEFIT, the critical parts are implemented in a binary library. For the unlimited FIRE version using linear terms, we also tried to partly optimize the implementation: we implemented the most time critical part, that is, the optimization part (Figure 2), as a C++ dynamic library for Java.

The results are shown in Table 2. The overall trend is that more accurate models take more time to be generated. The FIRE, RULEFIT and DIRTY methods seem to be more time consuming. For DIRTY, the data sets with multiple targets require heavier computing, while for a small number of targets the algorithm is quite fast.

Adding linear terms in FIRE further increases the average time usage. However, the effect depends on the data set. On some data sets, like SIGMEA REAL and CPU ACTIVITY, linear terms increase the accuracy without increasing the running time.

Usually, however, adding more base models (rules and linear terms) to an ensemble also increases the amount of computing needed. As mentioned in Section 3, the total number of added linear terms is the number of numeric descriptive attributes times the number of target attributes. This is probably a reason for the long running time of FIRE with linear terms and why the difference between the two unlimited FIRE versions is greater for multi-target data sets than for single-target ones. The large difference between the times of the unlimited versions of FIRE for data sets like META LEARNING and AUTO-PRICE is caused by trying a different number of τ parameter values for optimization (Figure 1, lines 9–10).

The limited FIRE versions use on average about a tenth of the time used by the unlimited ones. The main reason for the decrease is that only a small portion of the covariance matrix is computed, as discussed in Section 3.3. Moreover, only a small number of weights is changed during each iteration.

By optimizing the implementation, we can surely decrease the time usage of FIRE: The unlimited version with linear terms was one fourth faster even with the suboptimal Java/C++ library solution shown in the rightmost column of Table 2. Nevertheless, as illustrated by the usually at most moderate difference between FIRE and the probably much more optimized RULEFIT implementation on single-target problems, there might not be a lot of room for further optimization. Nevertheless, we should be able to speed up FIRE by limiting the number of possibilities covered. For example, we could try out a smaller number of τ values (Figure 1, line 5).

Data set	Tree	Random forest	Fire	Fire+ linear	Fire, Max 30	Fire+linear, Max 30	MtSmoti	Dirty	RuleFit	RuleFit+ linear	RegEnder	Fire+linear (C++)
Abalone	3.3	24.3	300.3	294.9	12.4	38.1	6.0	21.9	146.6	140.0	8.2	
Ailerons	2.9	10.1	258.4	315.4	5.6	13.0	15.9	157.6	52.0	52.7	15.2	
Auto-mpg	2.1	4.5	301.3	336.6	4.2	5.4	1.2	7.3	12.2	10.4	1.5	
Auto-price	1.7	2.9	92.5	5.6	3.2	3.2	1.2	42.6	3.8	3.6	0.8	
Breast cancer	1.9	3.8	44.8	49.6	17.7	17.3	0.7	14.4	5.9	5.8	1.5	
Census	9.8	185.7	531.0	551.1	43.5	47.4	44.0	84.0	1,131.7	1,133.1	108.5	
Cloud	1.6	2.8	3.8	138.4	2.7	15.3	0.6	13.0	2.5	2.1	0.6	
CPU activity	5.0	50.0	289.5	292.7	14.0	26.2	22.0	76.2	707.1	682.0	50.5	
Computer hw	2.0	3.3	56.0	5.6	2.7	3.9	1.1	13.1	6.0	5.6	1.0	
Delta-ailerons	3.5	37.4	269.2	295.4	16.2	21.7	5.0	15.0	247.0	230.5	10.4	
Diabetes	1.4	2.1	33.1	33.4	2.7	2.8	0.3	2.5	0.8	0.7	0.5	
Echocardiogram	1.7	3.1	23.1	24.2	3.5	3.6	0.5	18.1	3.5	3.4	0.6	
Housing	2.2	4.9	267.1	316.7	4.4	5.2	2.5	33.5	14.7	14.7	1.4	
Housing CA	10.3	162.1	428.1	351.0	36.4	63.2	353.1	77.0	1,875.1	1,814.4	83.5	
Kinematics	4.5	56.3	405.1	423.4	14.9	20.0	16.2	37.0	720.8	742.7	23.1	
Meta-data	2.5	3.9	3.9	3.8	4.3	4.0	2.5	33.9	10.6	10.7	4.3	
Pbc	2.2	4.9	54.8	64.7	7.7	9.1	2.4	28.3	13.3	13.4	1.4	
Pole Telecomm	8.6	50.4	353.8	463.7	31.8	48.0	59.0	1,386.5	488.5	496.3	423.3	
Pyrimidines	1.7	2.3	2.8	5.0	2.7	3.4	1.0	64.5	1.1	1.0	0.7	
Quake	2.4	12.9	119.0	131.1	18.1	18.8	2.1	4.7	53.8	53.9	1.5	
Sensory	2.1	4.7	31.2	31.3	6.3	5.9	1.1	13.1	28.2	28.2	1.6	
Servo	1.6	2.9	32.1	30.4	2.9	2.8	0.4	4.5	4.9	4.9	0.7	
Strike	2.2	5.6	16.9	16.1	16.9	16.9	1.2	8.4	16.0	16.1	2.1	
Veteran	1.5	2.8	156.1	172.9	13.7	17.0	0.5	9.3	2.6	2.6	0.6	
Average – single target	3.3	26.8	169.7	181.4	12.0	17.2	22.5	90.3	231.2	227.9	31.0	
Collembolan	1.9	6.1	92.1	162.6	6.6	7.7	5.0	367.0				91.0
Edm	1.3	2.0	43.8	58.2	3.3	3.9	2.2	70.2				40.4
Forest Kras	176.0	937.0	3,851.8	20,833.4	671.6	1,741.2	2,363.3	615,376.3				17,204.1
Forest Slivnica	9.8	56.9	371.7	1,047.7	19.3	26.1	222.6	11,365.3				479.9
Meta learning	1.5	2.2	50.9	698.5	13.4	17.2	1.9	1,218.9				293.6
Microarthropods	2.6	5.7	194.6	745.1	20.5	31.6	37.8	6,196.6				362.8
Sigmea real	1.5	5.4	28.9	30.9	2.9	3.6	2.0	7.7				15.0
Sigmea simulated	4.8	25.0	257.4	291.9	16.4	18.4	52.3	150.1				199.7
Solar flare	1.3	2.4	10.0	9.9	13.3	12.7	0.8	25.3				6.9
Vegetation	38.4	497.1	1,758.2	6,425.3	143.5	362.5	1,002.5	50,165.2				4,823.9
Water quality	2.2	14.2	437.6	829.6	35.7	38.9	7.8	294.4				467.8
Average – multi-target	22.0	141.3	645.2	2,910.5	86.0	205.8	336.2	62,294.3				2,180.5

Table 2: Running times in seconds for processing single-target and multi-target regression data sets.

5.5 Analysis on the Effect of Linear Terms on FIRE

In this section, we analyze the effect that linear terms have on the performance of FIRE. As noted in Sections 5.2 and 5.3, small FIRE models do not seem to benefit from the linear terms in the multi-target setting. In single-target domains and for unlimited FIRE, however, the effect is always positive. Especially for the small size limited models in single-target case, the effect is very significant as mentioned in Section 5.1. The reason for this phenomenon can be understood when we remember the definition of the linear terms in the multi-target model:

$$x_{(t,j)} = (0,\ldots,\underset{t-1}{0},\underset{t}{x_j},\underset{t+1}{0},\ldots,0).$$

The predictive power of a linear term clearly lowers with more targets and is the highest for single-target data. This is due to the fact that a linear term gives prediction only to a single target even if its effect on the ensemble size is the same as that of a rule. A rule, on the other hand, predicts all the targets at once and, thus, has a higher effect on the loss function. That is, in case of limited FIRE, each linear term occupies a slot of a rule but only has an effect corresponding to a proportion of $1/T$. In the unlimited FIRE we do not have such a contest between linear terms and rules during the optimization and, thus, the negative effect of adding a linear term is negligible.

The numbers of rules and linear terms presented in Tables 4 and 5 (Appendix D) support this line of reasoning. For the limited FIRE, the average number of linear terms lowers from 12% for single-target to 5% for multi-target. For the unlimited version, the values are respectively 2% and 24%. For the limited FIRE, the optimization process does not consider linear terms helpful in the multi-target case. Only a small number of linear terms is kept in the resulting model and they seem to have a negligible or even a negative effect on the accuracy. Moreover, apart from the WATER QUALITY data set, the data sets with most targets give rise to models with a greater portion of linear terms. Apparently, with many targets we need more linear terms to achieve a similar effect.

The results with different size limitations in Section 5.3 give an interesting view on the effect of linear terms on the models. In the multi-target case, only larger models benefit from adding the linear terms. The effect is negative for smaller models. The threshold in our case seems to be around the size limit of 50 terms. Clearly, these results further indicate that linear terms are of benefit only if we already have enough rules to cover the predictive problem well enough. With smaller limitations, they only take place away from the more powerful rules. However, for single-target rule ensembles the linear terms seem to help all the time as seen in Figure 5. A similar effect can be assumed for data sets with a small number of targets.

The issues presented above raise the question whether there is a better, more compressed way to include linear terms in the ensemble. We could, for instance, have a random combination of target dimensions covered for each linear term. It is an interesting question whether this would benefit the multi-target rule ensembles in the same way as the (single dimension) linear terms benefit the single-target rule ensembles.

We could also consider altering the normalization process for the linear terms described in Section 3.2. To add more effect to the linear terms, we could simply multiply the single nonzero prediction with the number of targets T. Intuitively this could bring the loss effect of a linear term to a level approximately equal to that of a rule. Unfortunately, in our experiments (not presented here) this seemed to stress linear terms too much and resulted in less accurate models.

Finally, it is not clear whether it is a good idea to count a linear term with the same weight as a rule when the model size is considered. After all, multiple linear terms can be united as shown

in Example 1 and are easier to interpret than a set of rules. This issue needs to be investigated in further work.

5.6 Summary of the Experimental Results

In our experiments, we first evaluated FIRE on single-target domains in order to show that our implementation of rule ensembles also works on standard regression problems. The results show that all versions of FIRE have accuracy that is comparable to the accuracy of the reference algorithms. The benefits of FIRE are even more evident when we take the model size into account: the most accurate models, random forests, are much larger.

Second and more important, we evaluated FIRE on multi-target domains. The results are somewhat similar to the ones obtained on single-target domains: random forests and the unlimited FIRE versions are more accurate than the limited FIRE, regression trees, and MTSMOTI. Nevertheless, the multi-task algorithm DIRTY performed much better now, being ranked right after the unlimited FIRE. As in the single-target case, the model size of MTSMOTI, regression trees, and the limited FIRE versions are statistically significantly smaller than the model size of the most accurate algorithms. Even though the difference in size between random forests and the unlimited FIRE is not statistically significant, the average size of a random forest is more than 300 times larger than the average size of FIRE models (with and without linear terms). In addition, although DIRTY performed well on average, its results were very unstable. Both the accuracy and computational resource usage varied highly from one data set to another.

Overall, the unlimited FIRE achieves good balance between accuracy and simplicity. Although it tends to generate somewhat less accurate models than the most accurate random forests, the size difference is very large. Therefore, we believe that the unlimited FIRE with linear terms is a very good choice for modeling multi-target regression problems.

Finally, the investigation of the influence of the maximum model size on the accuracy confirms that this parameter can be successfully used to control the accuracy-for-simplicity trade-off. The results show the general trend of larger models having better accuracy. The fact that the trend is more evident in the multi-target domains can be attributed to multi-target tasks being more complicated and demanding more complex models for maximal accuracy. In the case of linear terms, we can conclude that for larger (50 terms or more) multi-target rule ensembles adding linear terms should in general improve the accuracy. However, more importantly, we should rethink the way linear terms are created and handled in the multi-target environment, trying to achieve as great a benefit in a multi-target as in single-target setting.

6. Conclusions and Further Work

In many application areas there is a need for methods that can learn interpretable multi-target models, that is, models that predict several target attributes simultaneously. Rules are undoubtably one of the most interpretable model types. We have adopted the rule ensemble approach and generalized it to multi-target regression domains.

The initial implementation of our algorithm FIRE (Aho et al., 2009) was already able to learn multi-target regression rule ensembles. In this work, we have extended it so that in addition to rules, linear terms can also be added to the ensemble. The performance of the algorithm has also been significantly improved by modifying the normalization and optimization steps. We also include a much more in-depth experimental evaluation, including more data sets and more reference algorithms.

Our implementation has a simple parameter for limiting the size (the number of rules and linear terms) of the learned model. This enables us to trade accuracy for size (and interpretability) of learned models. We evaluated our algorithm with two parameter values: One that limits the number of rules to a maximum of 30, and one that has no size restrictions. We compared it with two other existing rule ensembles approaches RULEFIT and REGENDER, and to other similar multi-target prediction algorithms, namely regression trees, random forests, and model trees (MTSMOTI). In addition, we compared our multi-target algorithm with a recent multi-task algorithm DIRTY. We also investigated how the size limit and adding linear terms to the ensemble affect the accuracy.

In both, the single-target and multi-target settings, the unlimited FIRE with linear terms performed well, especially when the model size is taken into account. The model size limitation parameter can be used to tune the accuracy-for-simplicity trade-off. In sum, FIRE achieves a good balance of accuracy and simplicity in the context of multi-target regression.

Let us conclude with some ideas for further work. There are still some features of RULEFIT that have not yet been added to FIRE. Thus, a natural direction for further work is to add these features and study if they improve the performance of FIRE. This might include, for example, the use of tree ensembles based on importance-sampling. FIRE uses the gradient descent method for weight optimization and an ad-hoc approach to selecting the optimal τ value. We could combine both optimization problems into a single one. However, such a combined problem might no longer be convex and gradient descent optimization could get trapped in local optima. A possible solution would be to use metaheuristic optimization methods, such as differential evolution or ant-colony optimization instead.

On the other hand, it would be interesting to explore whether and how the recent multi-task optimization methods can be adapted to the multi-target environment. Moreover, our experiments suggest that we should further study the efficient application of linear terms to multi-target prediction. Finally, the automated exploration of the accuracy-for-simplicity trade-off by selecting an appropriate model size (the number of rules) deserves further attention.

Acknowledgments

We would like to thank the anonymous reviewers for insightful and helpful comments. We also want to thank Ali Jalali, Jerome H. Friedman and Annalisa Appice for the generous help with the implementations of their algorithms.

The work of T. Aho and T. Elomaa was supported by the Academy of Finland (through the research project REALM *Reactive learning and mining*). Moreover, the work of T. Aho was financially supported by Tampere Doctoral Programme in Information Science and Engineering.

The work of B. Ženko and S. Džeroski was supported by the Slovenian Research Agency (Grants P2-0103 and J2-2285), the European Commission (Grants ICT-2010-266722 and ICT-2011-287713), and Operation no. OP13.1.1.2.02.0005 financed by the European Regional Development Fund (85%) and the Ministry of Education, Science, Culture and Sport (15%).

Appendix A. Gradient Step Size

In this section, we present the reasoning behind the step size selection used in the FIRE gradient descent optimization procedure (Figure 2, line 17).

First of all, let us change the predictive function in Equation 2 to the following form:

$$f(x) = w_0\, avg + \sum_{i=1}^{M} w_i r_i(x) + \sum_{j=1}^{J}\sum_{k=1}^{K} w_{(j,k)} x_{(j,k)} = w\left(P_1^\mathsf{T}, \ldots, P_J^\mathsf{T}\right).$$

Here J is the number of targets and each P_j^T represents the predictions for the target dimension j:

$$P_j = \left(avg_j, r_1(x)_j, r_2(x)_j, \ldots \left[x_{(1,1)}\right]_j, \left[x_{(1,2)}\right]_j, \ldots\right).$$

In our case, the loss function is presented by Equation 5 as

$$L(x;w) = \frac{1}{2J}\sum_{j}^{J}(f_j(x) - y_j)^2 = \frac{1}{2J}\sum_{j}^{J}(wP_j^\mathsf{T} - y_j)^2.$$

Let us denote the weights by w_k and the gradients by g_k at the end of the iteration k (Figure 2, at the beginning of line 17). The step we are taking to change the weight is defined by $w_{k+1} = w_k - \beta g_k$, where β is the step size. Now, in the iteration k we want to minimize the one-dimensional loss function over step size

$$\Phi(\beta) = L(x; w_k - \beta g_k),$$

where

$$g_k^\mathsf{T} = \partial L/\partial w_k = \frac{1}{J}\sum_{j}^{J}\left(w_k P_j^\mathsf{T} - y_j\right)P_j^\mathsf{T}.$$

By finding the zero point of the derivative of the expected value

$$\mathbf{E}\,\Phi'(\beta) = \mathbf{E}\,\frac{1}{J}\sum_{j}^{J}\beta(g_k P_j^\mathsf{T})^2 - (w_k P_j^\mathsf{T} - y_j)g_k P_j^\mathsf{T}$$

we get the optimal step size β^* for minimizing the loss function Φ during each iteration k:

$$\beta^* = \frac{\mathbf{E}\,J\|g_k\|^2}{\mathbf{E}\,\sum_j^J\left(g_k P_j^\mathsf{T}\right)^2}. \tag{6}$$

Unfortunately, computing the precise value for each iteration is very costly. Thus, let us try to find some approximation. By using the Cauchy–Schwarz inequality we get the following upper bound for the numerator:

$$(gP_j^\mathsf{T})^2 \leq gg^\mathsf{T}P_jP_j^\mathsf{T}.$$

Now, noting that $\|g\|^2 = 0$ means a zero size step, we can write the following bound for the optimal step size β^*:

$$\beta^* \geq \frac{J}{\mathbf{E}\,\sum_j^J\|P_j(x)\|^2} = \beta_{\text{low}}.$$

Because of the normalization explained in Section 3.1 and Appendix B, we know that during the optimization $avg = 1$. Thus, we note that $\beta_{\text{low}} \leq 1$ always holds for the bound.

We could start with step size 1 and reduce the step size in a logarithmic fashion so that after a while we reach the lower bound. The optimal step size β^* depends on the gradient as seen in Equation 6. Reducing the step size like this would cause β to be near the optimal value most of time and end below it. However, in practice it seems that using β_{low} as a step size gives good results. Thus, we simply use this lower bound as a constant step size.

Appendix B. Justification for the Normalization Process

In this section, we give reasons for the decisions of the normalization process described in Section 3.1.

As we recall from Equation 2, our model is of the form

$$f(x) = w_0\, avg + \sum_{i=1}^{M} w_i r_i(x) + \sum_{j=1}^{J}\sum_{k=1}^{K} w_{(j,k)} x_{(j,k)}.$$

The linear terms in the last sum are optional and we concentrate on the rules part first.

First we rationalize why the first step, centering, in normalization presented in Section 3.1 is needed. We remember that our rules are transformed from a tree ensemble of form:

$$f(x) = \frac{1}{|D|}\sum_{i=1}^{|D|} d_i(x),$$

where $|D|$ is the number of trees in the ensemble. Here each tree prediction d_i in the ensemble is *global* in contrast to rule predictions r_i being *local*. That is, the tree prediction functions d_i give a prediction to all possible instances x, while rules predict only the subset they cover. Otherwise the rule function r_i equals zero. In other words, the tree prediction functions cover all the instances.

Initially, right after converting the trees to rules, we know that out of the M rules, exactly $|D|$ cover an arbitrary instance x. This consists of a rule from each of the trees in the tree ensemble. In this case, we can simply take an average of the predictive functions to get the overall prediction:

$$f(x) = \frac{1}{|D|}\sum_{i=1}^{M} r_i(x).$$

Now the rule predictions are used in the same scale in which they were created during the tree ensemble training.

Nevertheless, this is problematic if we omit a part of the initial rule set as is done in normalization. In this case, we do not know how many rules cover the given instance and, thus, can not simply take average of the covering instances. A simple solution to this problem is to remove the average offset that is included in all the predictions. That is, we replace each of the initial rules r'' with a zero-centered rule r', which is defined as

$$r'(x) = \begin{cases} r''(x) - avg & \text{if } x \text{ is covered and} \\ 0 & \text{otherwise.} \end{cases}$$

Now we can define a rule set equivalent to the initial one with

$$\begin{aligned} &\frac{1}{\text{nb. of covering rules}}\sum_{i=1}^{M} r_i''(x) \\ &= \frac{\text{nb. of covering rules}}{\text{nb. of covering rules}}\, avg + \sum_{i=1}^{M} r_i'(x) = avg + \sum_{i=1}^{M} r_i'(x). \end{aligned}$$

This new form allows us to do the weight optimization freely without caring about the number of covering rules:

$$f(x) = w_0\, avg + \sum_{i=1}^{M} w_i r_i'(x).$$

At the second stage of normalization, our aim is to equalize the rules with respect to the optimization problem presented in Equation 4:

$$\arg\min_{w} \sum_{(x,y)\in E} L(f(x),y) + \lambda \sum_{i=1}^{M} |w_i|^{\alpha}.$$

The optimization problem is not invariant to the scaling of rule predictions: If we scale the rule predictions as $r = br'; b > 1$, the corresponding weight will not be simply decreased as $w = w'/b$, because the regularization part on the right only includes weights and not rule predictions. As a result, the rules with smaller (absolute) predicted values are penalized more during optimization than the ones with larger predicted values. We would like to have all the rules and targets to have equal initial importance.

The obvious approach of setting a constant value to the base model target predictions is suboptimal for multi-target problems. Let us illustrate this with an example. We are given two target features y_1, y_2 that are highly inversely linearly dependent so that linear base model of type $(1, -5)$ would give high predictive accuracy. It is clear that setting 1 to the rule predictions for all targets would discard all the discovered information on relations between the targets stored in the rules.

Thus, we choose to scale each rule $r'(x)$ with a value that corresponds to its initial predictive size $\chi(r')$:

$$r = \frac{r'}{\chi}. \tag{7}$$

But how should we choose the exact value of χ?

To simplify the relations of target prediction ranges it is useful to bound the scaled predictions to some closed interval, for example, $[-1,1]$. In this case χ should be related to the largest predicted value of the rule r'. However, we also have to note the differing scales of the target attributes t'_t. If σ_t were the standard deviation of a normally distributed target attribute t'_t, dividing a zero-centered attribute by $2\sigma_t$ should put 95% of all values within the $[-1,1]$ interval.

Thus, when target prediction r'_m is the largest, we would use as the normalization factor χ in Equation 7 the value

$$\chi = \frac{r'_m}{2\sigma_m}.$$

More in detail, the index $m \in \{1, \ldots, T\}$ of the maximum target value r'_m of the rule r' is defined by:

$$m = \arg\max_{t} \left| \frac{r'_t}{2\sigma_t} \right|.$$

This way we make all the rules have equal maximal target prediction. We can now also give a strict bound to the predictions r_t: By the definition of m, after this second stage of normalization it holds that $r_m = 1$ and $|r_t| = |\sigma_m / r'_m r'_t / \sigma_t| \leq 1$ for all other targets t. Alternative choices for the normalization factor χ exist, but they may not behave as well in practice (Aho, 2012).

To sum up, at the second stage of normalization the target predictions r_t in a certain rule r are scaled by a factor χ that represents the the initial size of the predictive values of the particular rule. Thus, this stage roughly equalizes the rules before the optimization phase and affects the rules of the final model. After the two first normalization steps our rule predictions are of form:

$$r = \frac{r'}{\chi} = \frac{r'' - avg}{\chi}.$$

In addition to equalizing the initial importance of rules, we also have to equalize the effect of differing target scales during the optimization. Otherwise, targets with large scales would dominate the rule selection. This is done in the third stage of the normalization. However, clearly our intention is to use this normalization only temporarily during optimization. Otherwise the resulting model would not anymore be applicable to real world data. As usual, we do this simply by dividing the target attribute prediction values r_t by twice their standard deviations $2\sigma_t$:

$$r_t^* = \frac{r_t}{2\sigma_t} = \frac{r_t'}{2\sigma_t\chi} = \frac{r_t'' - avg_t}{2\sigma_t\chi}.$$

The rationalization on the normalization of the linear terms is similar to what was presented here for the rules. Nevertheless, linear terms are global in nature.

We recall that single linear term is defined as

$$x_{(t,j)} = (0, \ldots, \underset{t-1}{0}, \underset{t}{x_j}, \underset{t+1}{0}, \ldots, 0),$$

which depicts the influence of the descriptive attribute x_j on the target attribute x_t. We add linear terms for all possible combinations of numeric descriptive attributes and target attributes.

Unlike rules, linear terms are affected by *two* attributes and, thus, *two* attribute space scales: that of the j-th descriptive attribute space and that of t-th target space. In addition, linear terms with their single nonzero coordinate are multi-target only in principle. There is no sense in scaling with the maximum coordinate of linear terms as was done in the second normalization stage. These differences have to be taken into account.

We again shift the linear terms by the average $\overline{x_j''}$ of the j-th descriptive attribute:

$$x'_{(t,j)} = (0, \ldots, x_j'' - \overline{x_j''}, \ldots, 0).$$

Here x_j'' is the original attribute x_j value. However, we continue by normalizing the terms to the target attribute scale

$$x_{(t,j)} = x'_{(t,j)} \frac{\sigma_t}{\sigma_j}.$$

Here we note the effect of two separate attribute spaces. Linear terms normalized like this appear in the final rule ensemble model.

However, analogously to the third stage of rule normalization we also scale the target dimension space out temporarily:

$$x^*_{(t,j)} = \frac{x_{(t,j)}}{2\sigma_t} = \frac{x'_{(t,j)}}{2\sigma_j}.$$

This is, again, only intended to equalize the terms of different target attributes during the optimization procedure.

In (Aho et al., 2009) we externally normalized the data. It is worth noting that the normalization process illustrated in Section 3.1 can not be trivially reduced to the external normalization. Let us concentrate on a single-target model and denote with † the functions and weights that result from the optimization process after external normalization. We now have the following form for the external

normalization results:

$$f^{\dagger}(x^{\dagger}) = w_0^{\dagger} avg^{\dagger} + \sum_{i=1}^{M} w_i^{\dagger} r_i^{\dagger}(x^{\dagger}) \overset{avg^{\dagger}=0}{=} \sum_{i=1}^{M} w_i^{\dagger} r_i^{\dagger}\left(\frac{1}{2}(x-\overline{x}) \odot \sigma^{-1}\right)$$
$$= \sum_{i=1}^{M} w_i^{\dagger} I_i(x) \frac{r_i''(x) - avg}{2\sigma_i} = \sum_{i=1}^{M} \frac{w_i^{\dagger}}{2\sigma_i} r_i'(x).$$

Here $\odot$ is the coordinate-wise Hadamard product and σ^{-1} is a vector which consists of the inverses of standard deviations: $1/\sigma$. Moreover, the indicator function I_i equals 1 if r_i' covers the instance and zero otherwise. The third equality follows from the fact that our tree ensemble training algorithm is invariant to scaling and behaves similarly with normalized and unprocessed data.

We notice that the form on last row can be expressed in the form of internal normalization

$$f(x) = w_0\, avg + \sum_{i=1}^{M} w_i r_i'(x)$$

only if $avg = 0$, that is, the data is originally zero-centered. We realize that a lacking term affects the optimization of the remaining weights. Thus, in the general case transforming an externally normalized model to an equivalent internally normalized one is not trivially possible.

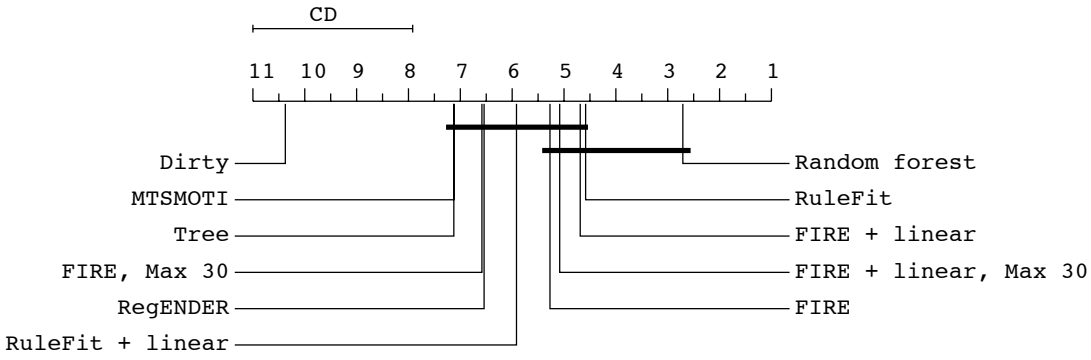
Appendix C. Comparison to RULEFIT

In this appendix, we study the differences between the FIRE and RULEFIT methods used in the experiments shown in Section 5.1. We used the RULEFIT settings that give the most accurate models as recommended by Friedman and Popescu (2008, 2004): these settings are presented in Section 4. However, an interesting question is prompted by the results in Section 5.1 and especially by Figure 3(a): Why does FIRE behave so much worse than RULEFIT on the single-target data? To what extent is this difference caused by the implementation and to what extent is it caused only by the different parameter settings implemented? In this section, we try to find an answer to these questions.

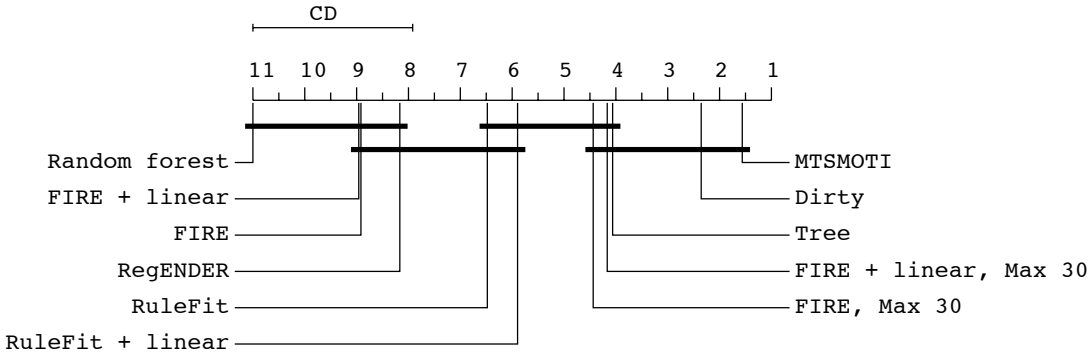
Figure 6 is analogous to Figure 3, the only difference is that now RULEFIT is using the parameter settings very similar to FIRE. As assumed, the behavior of the two algorithms is now much more alike. As seen in Figure 6(a), the accuracy of the unlimited FIRE versions is between those of the RULEFIT models. Surprisingly, however, FIRE now seems to gain more from linear terms while RULEFIT is more accurate without them.

There are several important RULEFIT features that could still be included in FIRE. The detailed list of differences is presented in Table 3. The most important difference is in the way the initial rule set is created. While FIRE (Figure 1, line 1) generates random forests, the best version of RULEFIT uses the best performing ISLE (Friedman and Popescu, 2003) tree ensembles. The major distinction here is that ISLE on each tree generation iteration takes into account the previously generated ensemble members when inducing the new tree. In this sense random forests are memoryless when compared with ISLE. As shown by Friedman and Popescu (2003), the difference in performance between ISLE and random forests may be very significant.

Another major difference is the robustness of RULEFIT. The used loss function and the linear terms are made robust against outliers. In addition, there are some minor differences. For example, for the best version of RULEFIT the size of each tree ensemble is larger and, instead of only leaves,



(a) RRMSE



(b) model size

Figure 6: Average ranks diagrams on *single-target* data with RULEFIT parameters similar to FIRE for *RRMSE* (a) and *model size* (b). Better algorithms are on the right-hand side, the ones that do not differ significantly by the Nemenyi test (p-value $= 0.05$) are connected with a horizontal bar. CD is the critical distance.

all tree nodes are turned into rules. This is possible because rule predictions are always set to 1 in the initial set of rules.

See Friedman and Popescu (2008, 2004) for more details about the parameter settings. We can conclude that we should be able to still improve the accuracy of FIRE by implementing some additional features present in RULEFIT.

Appendix D. Detailed Experimental Results

The detailed results discussed in Sections 5.1 and 5.2 are presented in Table 4 for single-target problems and in Table 5 for multi-target problems. In the table, DIRTY size is presented in number of nonzero weights as was done by Jalali et al. (2011). However, since small magnitude weights may disturb the view, we here also report average sizes where weights having absolute value under a threshold are removed. With thresholds $c \cdot \text{median}(B+S)$ where $c \in \{0, 0.01, 0.1, 0.3, 0.5\}$, we have the sizes 10.4, 10.2, 9.5, 8.5, 7.6 in single-target and 293, 264, 228, 198, 178 in multi-target cases. Thus, the amount of very small weights seems not to be very relevant. The amount of optimized weights altogether was on average 12.6 in single-target and 336.7 in multi-target.

SETTING	BEST RULEFIT	RULEFIT MOST SIMILAR TO FIRE	FIRE
LINEAR FUNC. TRIMMING	0.025	0.0	0.0
HUBER LOSS TRIMMING	0.9	(SQR LOSS) 1.0	(SQR LOSS) 1.0
MAX. NB OF RULES	5000	1000	ON AVG $\approx$700
MEAN MAX. NB. OF LEAVES	6	7	$\approx$7
INCENTIVE FACTOR	3*	(NOT USED) 1.0	(NOT USED) 1.0
TREE MEMORY FACTOR	0.01	(NOT USED) 0.0	(NOT USED) 0.0
TREE SAMPLING FRACTION	$1/5$	$1-1/e$	$1-1/e$
GD VALID. SET SIZE	3-FOLD XVAL	1/3	1/3
GD STEP SIZE	0.01*	0.01*	SEE APPENDIX A

Table 3: Setting differences for FIRE and RULEFIT in single-target experiments. The value is marked with an asterisk if the best value of RULEFIT was not mentioned in the original papers or if value similar to FIRE was not available. Default values were used in these cases.

	TREE		RANDOM FOREST		FIRE		FIRE + LINEAR		FIRE, MAX 30		FIRE + LINEAR, MAX 30		MTSMOTI		DIRTY		RULEFIT		RULEFIT + LINEAR		REGENDER	
DATA SET	RRMSE	#	RRMSE	#	RRMSE	#	RRMSE	# [%]	RRMSE	#	RRMSE	# [%]	RRMSE	#	RRMSE	#	RRMSE	#	RRMSE	# [%]	RRMSE	#
ABALONE	0.72	68	**0.67**	136,291	0.68	810	0.68	817 [1]	0.73	30	0.68	30 [23]	0.69	14	1.27	8	**0.67**	214	**0.67**	162 [2]	**0.67**	200
AILERONS	0.67	54	0.58	49,442	0.58	300	0.53	361 [9]	0.62	30	0.51	30 [77]	0.54	2	2.65	6	0.51	189	**0.50**	187 [3]	1.02	200
AUTO-MPG	0.45	16	**0.36**	14,310	0.37	780	0.37	768 [1]	0.39	30	0.37	30 [7]	0.46	11	1.73	7	**0.36**	164	**0.36**	116 [2]	0.39	200
AUTO-PRICE	0.49	9	0.35	5,554	**0.34**	629	0.36	120 [8]	0.43	30	**0.34**	30 [17]	0.39	6	1.02	15	0.38	125	0.38	125 [2]	0.37	200
BREAST CANCER	**0.96**	3	0.99	8,727	0.99	724	1.00	725 [0]	0.97	30	0.97	30 [0]	1.01	7	1.10	9	0.97	13	0.97	13 [0]	1.04	200
CENSUS	0.63	504	**0.55**	827,401	0.59	827	0.59	835 [1]	0.69	30	0.68	30 [10]	0.66	12	1.37	8	0.63	195	0.63	181 [2]	0.58	200
CLOUD	0.58	9	0.48	4,185	0.48	34	**0.42**	626 [1]	0.52	30	**0.42**	30 [10]	0.58	4	1.09	6	0.54	108	0.50	25 [8]	0.54	200
CPU ACTIVITY	0.19	317	**0.15**	270,473	0.18	718	0.17	730 [2]	0.24	30	0.22	30 [10]	0.19	6	1.06	12	0.16	257	0.20	163 [6]	0.18	200
COMPUTER HW	0.96	3	0.39	6,996	0.39	538	0.32	105 [6]	0.57	30	0.33	30 [13]	**0.22**	2	0.52	7	0.59	122	0.57	90 [6]	0.45	200
DELTA-AILERONS	0.60	106	**0.53**	210,808	0.54	600	0.54	549 [1]	0.59	30	0.55	30 [10]	0.57	16	0.87	5	0.55	217	0.55	148 [1]	0.95	200
DIABETES	1.02	3	0.89	1,610	0.89	385	0.89	387 [1]	**0.81**	30	0.84	30 [7]	0.99	1	2.47	2	0.88	22	0.90	22 [0]	1.05	200
ECHOCARDIOGRAM	**0.68**	3	0.76	4,472	0.85	579	0.85	537 [1]	0.86	30	0.87	30 [3]	0.76	3	1.08	9	0.78	87	0.78	86 [2]	0.92	200
HOUSING	0.43	32	**0.36**	18,864	0.38	379	0.38	777 [2]	0.45	30	0.44	30 [10]	0.47	4	1.18	13	0.37	190	0.37	175 [2]	0.39	200
HOUSING CA	0.51	745	**0.42**	782,382	0.54	745	0.52	827 [1]	0.63	30	0.58	30 [10]	0.60	13	1.44	8	0.48	257	0.47	202 [2]	0.47	200
KINEMATICS	0.72	290	**0.53**	318,707	0.63	722	0.63	619 [1]	0.74	30	0.72	30 [13]	0.77	4	2.83	8	0.58	220	0.58	205 [0]	0.54	200
META-DATA	1.03	1	**0.96**	5,613	0.99	54	1.00	0 [0]	0.99	30	1.00	0 [0]	1.00	15	0.98	21	0.99	66	0.99	66 [0]	1.59	200
PBC	0.94	6	**0.82**	13,687	0.83	748	0.83	758 [1]	**0.82**	30	**0.82**	30 [10]	0.84	2	1.62	18	**0.82**	130	**0.82**	133 [2]	0.84	200
POLE TELECOMM	0.17	377	0.20	96,380	0.30	757	0.30	783 [3]	0.38	30	0.38	30 [23]	0.26	21	0.97	30	0.19	123	0.19	125 [3]	**0.16**	200
PYRIMIDINES	0.84	6	0.77	1,971	0.81	26	0.82	199 [8]	0.81	26	0.89	8 [25]	1.77	2	1.15	26	0.66	45	**0.65**	26 [15]	0.71	200
QUAKE	**0.99**	3	1.03	71,100	1.03	91	1.03	91 [2]	1.01	30	1.01	30 [3]	1.00	2	24.7	3	1.08	0	1.08	0 [0]	1.01	200
SENSORY	0.94	6	**0.85**	17,972	**0.85**	128	**0.85**	128 [0]	**0.85**	30	**0.85**	30 [0]	0.91	16	6.29	11	**0.85**	167	**0.85**	167 [0]	0.89	200
SERVO	0.42	11	**0.31**	5,555	0.38	436	0.38	436 [0]	0.42	30	0.42	30 [0]	0.55	7	0.89	4	0.38	188	0.38	188 [0]	**0.31**	200
STRIKE	0.98	12	0.93	23,379	0.95	404	0.96	410 [1]	0.93	30	0.93	30 [0]	0.95	2	1.00	6	**0.88**	135	**0.88**	140 [0]	0.98	200
VETERAN	0.99	2	**0.92**	4,872	0.93	418	0.93	417 [0]	0.94	30	0.93	30 [3]	0.93	4	0.94	7	0.96	54	0.95	51 [2]	1.16	200
AVERAGE	0.68	108	**0.62**	120,865	0.65	493	0.64	500 [2.1]	0.68	30	0.66	28 [12]	0.71	7.3	2.51	10	0.64	137	0.63	116 [2.5]	0.72	200

Table 4: Comparison of *RRMSE* for *single-target* regression. In each row, the smallest error is typeset in bold. Size is given either as the number of rules and linear terms or as the number of terminal nodes in trees. Besides the model sizes (#), we also give in brackets the percentage of linear terms within the total model size ([%] if present). The final row gives the average errors and rule set sizes over all data sets.

DATA SET	TREE		RANDOM FOREST		FIRE		FIRE + LINEAR		FIRE, MAX 30		FIRE + LINEAR, MAX 30		MTSMOTI		DIRTY	
TARGET ATTRIBUTES	RRMSE	#	RRMSE	#	RRMSE	#	RRMSE	# [%]	RRMSE	#	RRMSE	# [%]	RRMSE	#	RRMSE	#
COLLEMBOLAN	0.97	3	0.92	13,145	0.93	438	0.93	397 [15]	0.96	30	0.95	30 [3]	1.04	16	**0.62**	141
SPECIES-NB	0.98		0.94		0.95		0.95		0.98		0.97		1.01		**0.37**	
ABUD-TOTAL	0.96		**0.93**		0.94		0.94		0.97		0.95		1.10		1.04	
ABUD-F.QUAD	0.96		0.89		0.90		0.90		0.94		0.93		1.01		**0.44**	
EDM	0.72	11	0.69	2,923	**0.68**	676	**0.68**	708 [5]	0.70	30	0.70	30 [3]	0.85	2	0.99	32
D-FLOW	0.68		0.66		0.66		**0.65**		0.71		0.71		0.92		0.99	
D-GAP	0.75		0.71		0.71		0.71		**0.69**		**0.69**		0.77		0.99	
FOREST KRAS	0.62	1,142	0.55	1,530,421	0.65	830	0.64	2,590 [68]	0.71	30	0.71	30 [0]	0.69	10	**0.21**	1,760
CC	0.53		**0.47**		0.56		0.54		0.61		0.61		0.57		0.48	
FSH	0.49		0.42		0.53		0.52		0.59		0.60		0.59		**0.18**	
DELVEG	0.53		**0.47**		0.55		0.54		0.61		0.61		0.56		**0.47**	
VPV1-HMX	0.60		0.52		0.63		0.62		0.70		0.71		0.68		**0.28**	
VPV1-H99	0.58		0.51		0.62		0.61		0.70		0.71		0.68		**0.22**	
VPV1-H95	0.57		0.50		0.61		0.60		0.69		0.70		0.67		**0.20**	
VPV1-H75	0.57		0.51		0.60		0.60		0.69		0.69		0.67		**0.16**	
VPV1-H50	0.60		0.54		0.63		0.63		0.71		0.71		0.70		**0.13**	
VPV1-H25	0.66		0.60		0.70		0.69		0.76		0.76		0.75		**0.09**	
VPV1-H10	0.76		0.69		0.78		0.78		0.83		0.83		0.82		**0.06**	
VPV1-H05	0.83		0.76		0.83		0.83		0.87		0.87		0.86		**0.04**	
FOREST SLIVNICA	0.54	321	0.49	227,198	0.50	618	**0.47**	872 [25]	0.58	30	0.58	30 [10]	0.51	4	0.49	298
HEIGHT	0.56		0.51		0.53		**0.49**		0.61		0.61		0.52		0.90	
COVER	0.52		0.48		0.47		0.45		0.54		0.55		0.50		**0.08**	

Continued on the next two pages.

Table 5: Comparison of *RRMSE* for *multi-target* regression. For each data set, we first give the *average RRMSE over all targets*, and then the *RRMSE for each target separately*. In each row, the smallest error is typeset in bold. Size is given either as the number of rules and linear terms or as the number of terminal nodes in trees. Besides the model sizes (#), we also give the percentage of linear terms within the total model size in brackets ([%] if present). The two final rows give the average RRMSEs over all targets, over data set target averages and average model size over all data sets.

DATA SET	TREE		RANDOM FOREST		FIRE		FIRE + LINEAR		FIRE, MAX 30		FIRE + LINEAR, MAX 30		MTSMOTI		DIRTY	
TARGET ATTRIBUTES	RRMSE	#	RRMSE	#	RRMSE	#	RRMSE	# [%]	RRMSE	#	RRMSE	# [%]	RRMSE	#	RRMSE	#
META LEARNING	1.36	2	0.92	1,415	**0.86**	464	0.88	472 [38]	0.99	30	0.96	30 [0]	1.20	2	1.23	10
LTREE	1.47		0.93		**0.85**		0.87		0.97		0.93		1.20		1.09	
C50-RULES	1.46		0.92		**0.82**		0.84		0.97		0.93		1.17		1.10	
LINDISCR	1.12		0.87		**0.79**		**0.79**		0.94		0.91		0.96		1.35	
MLCIB1	1.49		0.95		**0.88**		0.89		1.02		0.99		1.28		1.25	
MLCNB	1.29		**0.92**		**0.92**		0.96		0.99		0.97		1.08		1.41	
RIPPER	1.33		0.93		**0.81**		0.82		0.85		0.83		1.13		1.19	
CLEM-RBFN	1.11		0.89		**0.88**		0.89		1.06		1.05		1.20		1.30	
C50-TREE	1.47		0.93		**0.85**		**0.85**		0.97		0.94		1.24		1.15	
CLEM-MLP	1.22		0.94		**0.93**		0.96		1.07		1.07		1.54		1.32	
C50-BOOST	1.51		0.95		**0.87**		**0.87**		1.00		0.96		1.20		1.13	
MICROARTHROPODS	0.77	52	0.74	12,411	0.75	642	0.75	992 [42]	0.85	30	0.85	30 [13]	0.83	18	**0.63**	417
ACARI	0.76		**0.71**		0.72		0.73		0.82		0.82		0.79		0.96	
COLLEMBOLAN	**0.74**		**0.74**		**0.74**		**0.74**		0.81		0.82		0.75		0.92	
SH-BIODIV	0.81		**0.75**		0.78		0.78		0.90		0.90		0.94		**0.00**	
SIGMEA REAL	**0.61**	12	0.65	22,506	0.69	147	0.65	209 [4]	0.74	14	0.72	30 [10]	0.62	7	1.00	8
MFO	**0.62**		0.67		0.74		0.72		0.75		0.77		0.64		0.99	
MSO	0.61		0.62		0.64		**0.59**		0.73		0.66		**0.59**		1.01	
SIGMEA SIMULATED	0.03	146	**0.02**	78,750	0.04	658	0.03	676 [3]	0.08	30	0.08	30 [17]	0.05	17	1.09	22
DISP-RATE	0.03		**0.02**		0.04		0.03		0.09		0.09		0.05		1.06	
DISP-SEEDS	0.03		**0.02**		0.03		0.03		0.08		0.07		0.04		1.12	
SOLAR FLARE	0.99	2	1.05	3,974	1.00	74	1.00	74 [0]	1.00	30	1.00	30 [0]	1.02	14	**0.93**	30
C-CLASS	0.99		1.03		1.00		1.00		1.00		1.00		1.03		**0.97**	
M-CLASS	**0.97**		1.04		0.99		0.99		0.98		0.98		1.00		**0.97**	
X-CLASS	1.01		1.07		1.01		1.01		1.03		1.03		1.04		**0.84**	

Table 5: Continued from the previous page and continued on the next page.

DATA SET	TREE		RANDOM FOREST		FIRE		FIRE + LINEAR		FIRE, MAX 30		FIRE + LINEAR, MAX 30		MTSMOTI		DIRTY	
TARGET ATTRIBUTES	RRMSE	#	RRMSE	#	RRMSE	#	RRMSE	# [%]	RRMSE	#	RRMSE	# [%]	RRMSE	#	RRMSE	#
VEGETATION	0.82	349	**0.72**	1,102,508	0.80	826	0.79	1,541 [46]	0.88	30	0.88	30 [0]	0.89	8	**0.32**	715
NUMBER-SPP	0.80		0.67		0.74		0.73		0.87		0.88		0.94		**0.02**	
DEM	0.68		0.52		0.65		0.63		0.78		0.79		0.81		**0.45**	
TWI	0.70		0.60		0.67		0.65		0.74		0.74		0.80		**0.13**	
SOLAR	0.99		0.97		0.99		0.98		0.99		0.99		0.99		**0.17**	
THINVK	0.96		**0.88**		0.94		0.94		0.98		0.98		0.96		**0.88**	
THK	0.96		**0.88**		0.94		0.94		0.98		0.98		0.96		**0.88**	
EFF-RAIN	0.68		**0.53**		0.66		0.64		0.78		0.79		0.81		0.57	
MINT-JUL	0.72		0.58		0.69		0.67		0.81		0.82		0.83		**0.29**	
MAX-FEB	0.68		0.53		0.67		0.65		0.82		0.83		0.85		**0.14**	
GRND-DPTH	0.81		0.71		0.78		0.77		0.86		0.87		0.89		**0.00**	
SALINITY	0.94		0.89		0.92		0.92		0.97		0.97		0.97		**0.00**	
WATER QUALITY	0.96	5	**0.90**	41,568	0.94	801	0.93	954 [19]	0.94	30	0.94	30 [0]	0.96	8	0.93	224
CLAD-SP	0.99		**0.94**		0.95		0.95		0.96		0.96		0.98		0.96	
GONG-INC	1.00		**0.97**		1.00		0.99		0.99		0.99		1.00		**0.97**	
OEDO-SP	1.00		0.94		0.96		0.95		0.97		0.97		0.99		**0.88**	
TIGE-TEN	0.95		**0.88**		0.93		0.91		0.93		0.93		0.95		**0.88**	
MELO-VAR	0.98		**0.90**		0.93		0.93		0.95		0.95		0.97		0.95	
NITZ-PAL	0.90		**0.83**		0.87		0.86		0.85		0.85		0.89		0.92	
AUDO-CHA	0.98		0.96		0.98		0.98		0.97		0.97		0.98		**0.82**	
ERPO-OCT	0.97		**0.91**		0.94		0.94		0.94		0.94		0.97		0.95	
GAMM-FOSS	0.93		**0.80**		0.83		0.83		0.89		0.89		0.91		0.91	
BAET-RHOD	0.97		**0.89**		0.95		0.94		0.95		0.95		0.96		0.98	
HYDRO-SP	0.98		**0.90**		0.95		0.94		0.95		0.95		0.97		0.96	
RHYA-SP	0.96		0.90		0.92		0.92		0.94		0.93		0.94		**0.85**	
SIMU-SP	0.99		**0.94**		0.98		0.97		0.99		0.99		1.00		0.97	
TUBI-SP	0.89		**0.84**		0.90		0.90		0.87		0.88		0.91		0.97	
AVERAGE – TARGETS	0.87		0.75		0.78		0.78		0.84		0.84		0.88		**0.71**	
AVERAGE – DATA SETS	0.76	185.9	**0.70**	276,075	0.71	561.3	**0.70**	862.3 [24]	0.77	28.5	0.76	30.0 [5.1]	0.79	9.6	0.77	293.3

Table 5: Continued from the previous two pages.

References

Timo Aho. *Steps on Multi-Target Prediction and Adaptability to Dynamic Input*. PhD thesis, Tampere University of Technology, Department of Software Systems, Tampere, Finland, 2012.

Timo Aho, Bernard Ženko, and Sašo Džeroski. Rule ensembles for multi-target regression. In Wei Wang, Hillol Kargupta, Sanjay Ranka, Philip S. Yu, and Xindong Wu, editors, *Proceedings of the Ninth IEEE International Conference on Data Mining (ICDM 2009)*, pages 21–30. IEEE Computer Society, 2009.

Annalisa Appice and Sašo Džeroski. Stepwise induction of multi-target model trees. In Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenić, and Andrzej Skowron, editors, *Proceedings of the 18th European Conference on Machine Learning (ECML 2007)*, LNCS, pages 502–509. Springer, 2007.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

Arthur Asuncion and David J. Newman. UCI machine learning repository, 2011. URL `http://archive.ics.uci.edu/ml/`.

Steffen Bickel, Jasmina Bogojeska, Thomas Lengauer, and Tobias Scheffer. Multi-task learning for HIV therapy screening. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, AICPS, pages 56–63. ACM, 2008.

Hendrik Blockeel. *Top-down Induction of First Order Logical Decision Trees*. PhD thesis, Katholieke Universiteit Leuven, Department of Computer Science, Leuven, Belgium, 1998.

Hendrik Blockeel and Jan Struyf. Efficient algorithms for decision tree cross-validation. *Journal of Machine Learning Research*, 3:621–650, 2002.

Hendrik Blockeel, Luc De Raedt, and Jan Ramon. Top-down induction of clustering trees. In Jude W. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning (ICML 1998)*, pages 55–63. Morgan Kaufmann, 1998.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

Olivier Chapelle, Pannagadatta Shivaswamy, Srinivas Vadrevu, Kilian Weinberger, Ya Zhang, and Belle Tseng. Boosted multi-task learning. *Machine Learning*, 85(1):1–25, 2010.

Delve. Data for evaluating learning in valid experiments, 2011. URL `http://www.cs.toronto.edu/~delve/index.html`.

Krzysztof Dembczyński, Wojciech Kotłowski, and Roman Słowiński. Solving regression by learning an ensemble of decision rules. In Leszek Rutkowski, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada, editors, *Proceedings of the Ninth International Conference on Artificial Intelligence and Soft Computing (ICAISC 2008)*, LNCS, pages 533–544. Springer, 2008.

Krzysztof Dembczyński, Wojciech Kotłowski, and Roman Słowiński. Maximum likelihood rule ensembles. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, AICPS, pages 224–231. ACM, 2008.

Damjan Demšar, Marko Debeljak, Claire Lavigne, and Sašo Džeroski. Modelling pollen dispersal of genetically modified oilseed rape within the field. In *Abstracts, the 90th ESA Annual Meeting*, page 152. The Ecological Society of America, 2005.

Damjan Demšar, Sašo Džeroski, Thomas Larsen, Jan Struyf, Jørgen Axelsen, Marianne Bruus Pedersen, and Paul Henning Krogh. Using multi-objective classification to model communities of soil microarthropods. *Ecological Modelling*, 191(1):131–143, 2006.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

Thomas Dietterich. Ensemble methods in machine learning. In Josef Kittler and Fabio Roli, editors, *Proceedings of the First International Workshop on Multiple Classifier Systems (MCS 2000)*, LNCS, pages 1–15. Springer, 2000.

Sašo Džeroski, Ljupčo Todorovski, and Hendrik Blockeel. Relational ranking with predictive clustering trees. In *Proceedings of the Workshop on Active Mining (in ICDM 2002)*, pages 9–15. IEEE Computer Society, 2002.

Sašo Džeroski, Andrej Kobler, Valentin Gjorgjioski, and Panče Panov. Using decision trees to predict forest stand height and canopy cover from LANDSAT and LIDAR data. In Klaus Tochtermann and Arno Scharl, editors, *Proceedings of the 20th International Conference on Informatics for Environmental Protection (EnviroInfo 2006)*, pages 125–133. Shaker, 2006.

Sašo Džeroski, Damjan Demšar, and Jasna Grbović. Predicting chemical parameters of river water quality from bioindicator data. *Applied Intelligence*, 13(1):7–17, 2000.

Sašo Džeroski, Nathalie Colbach, and Antoine Messéan. Analysing the effect of field character on gene flow between oilseed rape varieties and volunteers with regression trees. In Antoine Messéan, editor, *Proceedings of the Second International Conference on Co-existence between GM and non-GM based Agricultural Supply Chains*, pages 207–211. Agropolis Productions, 2005.

Peter Flach and Nada Lavrač. Rule induction. In Michael R. Berthold and David J. Hand, editors, *Intelligent Data Analysis*, pages 229–267. Springer, 2003. Second edition.

Jerome H. Friedman and Bogdan E. Popescu. Importance sampled learning ensembles. Technical report, Stanford University, Department of Statistics, Stanford, CA, 2003.

Jerome H. Friedman and Bogdan E. Popescu. Gradient directed regularization for linear regression and classification. Technical report, Stanford University, Department of Statistics, Stanford, CA, 2004.

Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. Technical report, Stanford University, Department of Statistics, Stanford, CA, 2005.

Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954, 2008.

Valentin Gjorgjioski, Sašo Džeroski, and Matt White. Clustering analysis of vegetation data. Technical Report 10065, Jožef Stefan Institute, Ljubljana, Slovenia, 2008.

Nitin Indurkhya and Sholom M. Weiss. Solving regression problems with rule-based ensemble classifiers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2001)*, pages 287–292. ACM, 2001.

Ali Jalali, Pradeep Ravikumar, Sujay Sanghavi, and Chao Ruan. A dirty model for multi-task learning. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS 2010)*, pages 964–972. MIT Press, 2011.

Minwoo Jeong and Gary Geunbae Lee. Multi-domain spoken language understanding with transfer learning. *Speech Communication*, 51(5):412–424, 2009.

Christian Kampichler, Sašo Džeroski, and Ralf Wieland. Application of machine learning techniques to the analysis of soil ecological data bases: relationships between habitat features and collembolan community characteristics. *Soil Biology and Biochemistry*, 32(2):197–209, 2000.

Aram Karalič. Employing linear regression in regression tree leaves. In B. Neumann, editor, *Proceedings of the Tenth European Conference on Artificial intelligence (ECAI 1992)*, pages 440–441. John Wiley & Sons, 1992.

Aram Karalič and Ivan Bratko. First order regression. *Machine Learning*, 26(2-3):147–176, 1997.

Dragi Kocev, Celine Vens, Jan Struyf, and Sašo Džeroski. Ensembles of multi-objective decision trees. In Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenić, and Andrzej Skowron, editors, *Proceedings of the 18th European Conference on Machine Learning (ECML 2007)*, LNCS, pages 624–631. Springer, 2007.

Qi Liu, Qian Xu, Vincent W. Zheng, Hong Xue, Zhiwei Cao, and Qiang Yang. Multi-task learning for cross-platform siRNA efficacy prediction: an in-silico study. *BMC Bioinformatics*, 11(1): 181–196, 2010.

Karim Lounici, Massimiliano Pontil, Alexandre B. Tsybakov, and Sara A. van de Geer. Taking advantage of sparsity in multi-task learning. In *Proceedings of the 22nd Conference on Learning Theory (COLT 2009)*, 2009.

Ryszard S. Michalski. On the quasi-minimal solution of the general covering problem. In *Proceedings of the Fifth International Symposium on Information Processing (FCIP 1969)*, volume A3, Switching Circuits, pages 125–128, 1969.

Shibin Parameswaran and Kilian Weinberger. Large margin multi-task metric learning. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS 2010)*, pages 1867–1875. MIT Press, 2011.

Beau Piccart, Jan Struyf, and Hendrik Blockeel. Empirical asymmetric selective transfer in multi-objective decision trees. In T. Horvath, J.-F. Boulicaut, and M. Berthold, editors, *Proceedings of the Eleventh International Conference on Discovery Science (DS 2008)*, LNAI, pages 64–75. Springer, 2008.

Ross J. Quinlan. Learning with continuous classes. In A. Adams and L. Sterling, editors, *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence (AI 1992)*, pages 343–348. World Scientific, 1992.

Alain Rakotomamonjy, Rémi Flamary, Gilles Gasso, and Stéphane Canu. $\ell_p - \ell_q$ penalty for sparse linear and sparse multiple kernel multitask learning. *IEEE Transactions on Neural Networks*, 22 (8):1307–1320, 2011.

StatLib. Data sets archive, 2011. URL `http://stat.cmu.edu/datasets/`.

Daniela Stojanova. Estimating forest properties from remotely sensed data by using machine learning. Master's thesis, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, 2009.

Jan Struyf and Sašo Džeroski. Constraint based induction of multi-objective regression trees. In F. Bonchi and J. Boulicaut, editors, *Proceedings of the Fourth International Workshop on Knowledge Discovery in Inductive Databases (KDID 2005)*, LNCS, pages 222–233. Springer, 2006.

Einoshin Suzuki, Masafumi Gotoh, and Yuta Choki. Bloomy decision tree for multi-objective classification. In Luc De Raedt and Arno Siebes, editors, *Proceedings of the Fifth European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2001)*, LNCS, pages 436–447. Springer, 2001.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

Luís Torgo. Regression DataSets, 2011. URL `http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html`.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, NY, 1995.

Yong Wang and Ian H. Witten. Inducing model trees for continuous classes. In *Proceedings of the Ninth European Conference on Machine Learning (ECML 1997) Poster Papers*, 1997.

Weka. Collections of datasets, 2011. URL `http://www.cs.waikato.ac.nz/~ml/weka/index_datasets.html`.

Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A comparison of optimization methods and software for large-scale L1-regularized linear classification. *Journal of Machine Learning Research*, 11:3183–3234, 2010.

Bernard Ženko. *Learning predictive clustering rules*. PhD thesis, University of Ljubljana, Faculty of computer and information science, Ljubljana, Slovenia, 2007.

Bernard Ženko and Sašo Džeroski. Learning classification rules for multiple target attributes. In Takashi Washio, Einoshin Suzuki, Kai Ming Ting, and Akihiro Inokuchi, editors, *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008)*, LNCS, pages 454–465. Springer, 2008.