Better Short than Greedy: Interpretable Models through Optimal Rule Boosting

Mario Boley* Simon Teshuva† Pierre Le Bodic‡ Geoffrey I. Webb§

Abstract

Rule ensembles are designed to provide a useful trade-off between predictive accuracy and model interpretability. However, the myopic and random search components of current rule ensemble methods can compromise this goal: they often need more rules than necessary to reach a certain accuracy level or can even outright fail to accurately model a distribution that can actually be described well with a few rules. Here, we present a novel approach aiming to fit rule ensembles of maximal predictive power for a given ensemble size (and thus model comprehensibility). In particular, we present an efficient branch-and-bound algorithm that optimally solves the per-rule objective function of the popular second-order gradient boosting framework. Our main insight is that the boosting objective can be tightly bounded in linear time of the number of covered data points. Along with an additional novel pruning technique related to rule redundancy, this leads to a computationally feasible approach for boosting optimal rules that, as we demonstrate on a wide range of common benchmark problems, consistently outperforms the predictive performance of boosting greedy rules.

1 Introduction

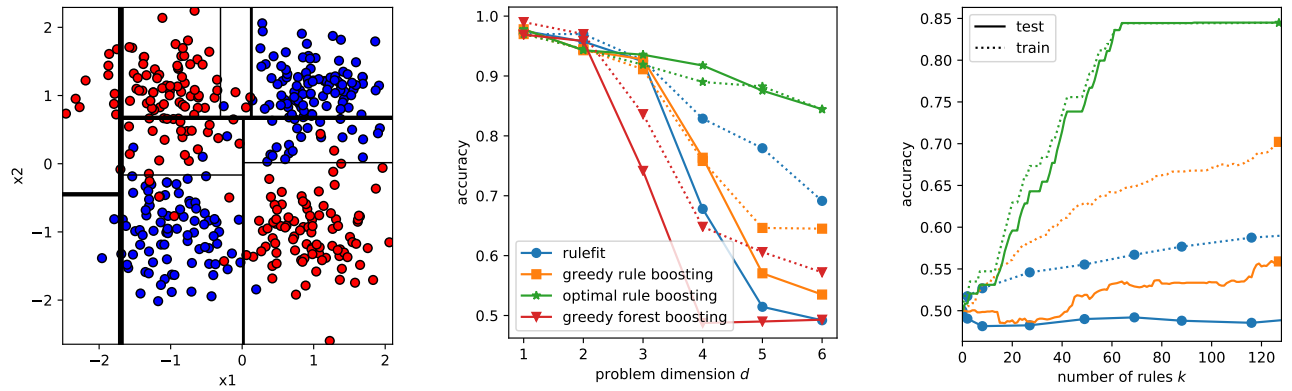
Rule learners are designed to deliver models that are interpretable and at the same time have a predictive performance that is competitive with complex tree ensembles. In particular, the gradient boosting approach provides a theoretically well-founded framework to combine simple prediction rules into powerful additive ensembles. However, the greedy and random search techniques that are traditionally employed to fit ensemble members can compromise model comprehensibility or, even worse, outright fail to adequately learn a distribution that can be described well with relatively few rules. This is because heuristically found rules tend to not fully capture higher order feature interactions, at least not in the simplest way possible (see Fig. 1). Consequently, a greater number of rules is required to achieve a certain predictive performance.

*Monash University, mario.boley@monash.edu
†Monash University, simon.teshuva@monash.edu
‡Monash University, pierre.lebodic@monash.edu
§Monash University, geoff.webb@monash.edu

As a remedy, this paper presents an optimal rule fitting procedure as base learner for gradient boosting and, contrary to the traditionally raised concerns, shows that: (i) *finding optimal rules in a boosting iteration is computationally feasible*—at least on common benchmark problems when targeting comprehensible ensembles with a small number of rules, and (ii) *optimal rules consistently outperform greedily found rules in terms of their predictive performance*—an effect that is again more pronounced for smaller ensemble sizes. Thus, the proposed algorithm is an important extension to the gradient boosting toolbox, in particular for fitting small interpretable rule ensembles.

1.1 Related Work Most predictive rule learning systems are *stagewise fitting* algorithms that identify individual rules one at a time based on their ability to improve the predictive performance of previously fixed rules. To achieve this, early *sequential coverage* (or *separate-and-conquer*) algorithms [see survey 12] simply remove data points covered by already selected rules before fitting subsequent stages, which results in rule lists that represent nested IF-ELSEIF-...-ELSE structures. These structures are typically hard to interpret because the effect of an individual rule has to be understood in the context of the preceding rules.

Additive rule ensemble algorithms instead produce flat sets of IF-THEN-rules, the outputs of which are simply added up to find the prediction for a given data point. Thus, individual rule effects on a prediction can be assessed in isolation. Examples for this approach are recent methods based on greedy sub-modular optimization [15, 22], earlier local pattern based methods [see 13], as well as *RuleFit* [11]. These methods build additive ensembles by performing a sub-selection from a set of candidate rules that are obtained in an initial pre-processing step. *RuleFit* is a particularly popular approach that overcomes the stagewise fitting paradigm and instead globally optimizes a sparse coefficient vector for the candidate rules. However, a general downside of fixing a set of pre-processed candidate rules is that this set might not contain optimal building blocks (e.g., the *RuleFit* candidates are based on greedy tree ensembles).



(a) greedy tree cuts, $d = 2$, $n = 400$ (b) acc. for var. d, $n = 100{\times}2^d$, $k = 2^d$ (c) acc. for var. k, $d = 6$, $n = 6400$

Figure 1: Weakness of heuristic rules in *noisy parity problem* defined by latent cluster centers $C \sim \text{Unif}(\{-1, 1\}^d)$, observed features $X \mid C \sim \text{Norm}(C, \sigma^2 I_d)$, and target variable $Y \mid C = \prod_{i=1}^{d} C_i$. Greedy cuts lead to sub-optimal partitioning already for $d = 2$ (a). Since there is no association between the target variable and any proper subset of features variables, i.e., $P(Y \mid X_{i_1}, \ldots, X_{i_l}) = P(Y)$ for all $\{X_{i_1}, \ldots, X_{i_l}\} \subset \{X_1, \ldots, X_d\}$, the first $d-1$ greedy choices are dictated by sample properties that are non-representative for the underlying distribution. In contrast, gradient boosting ensembles with optimal rules approximate with high probability the 0/1-risk minimizing 2^d rules "$(-1)^{|N|}$ if $\bigwedge_{i \in N} X_i \leq 0 \bigwedge_{i \in P} X_i > 0$" for all $N \subset \{1, \ldots, d\}$ and $P = \{1, \ldots, d\} \setminus N$. Correspondingly, optimal rules retain non-trivial accuracy for growing d whereas the performance of heuristic rules quickly deteriorates to the level of random guessing (b). Importantly, the performance of the heuristic ensembles is not improved for $n \to \infty$ but only with a substantial increase of the number of ensemble members k (c).

An alternative line of methods [6, 7, 8] produces additive rule ensembles by adopting the *gradient boosting* framework [5, 10]. This is again a stagewise process, in which new rules are fitted based on their effect on the training loss when their output is added to the prediction scores of the previously fixed rules. This approach is universally applicable to various kinds of learning problems—provided that the predictive performance for an individual training point can be quantified by a differentiable loss function. Moreover, it does not require a bounded candidate set and instead optimizes in each stage a statistically sound objective function over all available conjunctive queries.

Notably though, current rule boosting methods inherited from their sequential coverage ancestors the heuristic search methods for fitting individual rules, i.e., greedy, beam, and stochastic search. Originally, this preference was not only motivated by computational efficiency but also to avoid overfitting (through "oversearching" [see 12, Sec. 4.1.3]). Given that boosting solves this issue in a more principled way, the exact optimization of the rule objective now appears unambiguously desirable—if the computational concern can be addressed.

While the statistical learning literature tends to avoid solving hard discrete optimization problems, the knowledge discovery and data mining literature often suggests to solve hard rule discovery problems exactly by exhaustive branch-and-bound search. Here, the conjunctive rule lattice is searched by iteratively augmenting conjunctions with further conditions/propositions (branch), but augmentations are skipped when they cannot lead to an improvement over the currently discovered optimum (bound). While this approach has a non-polynomial worst-case time complexity, there are a number of techniques that make it practically applicable to many datasets.

One idea, introduced in the *OPUS (Optimized Pruning for Unordered Search) framework* [20, 21], is to only check an augmentation if it successfully passed the pruning check on the parent level of the current search node. This technique leverages the monotonicity of the bounding function and usually achieves speed-ups proportional to the number of available propositions.

A further development is the introduction of *tight bounding functions* (also "tight optimistic estimators") [14, 17] that bound the value of all possible refinements of a conjunction by identifying the optimal sub-selection of the data points selected by that conjunction. While this approach disregards whether this sub-selection can be described by an actual refinement, it is much more effective than simple term-wise upper bounding (that one

obtains by more straightforward adaptions of support-based pruning) and, compared to those, reliably leads to orders of magnitude speed-up [16].

A final crucial technique is to eliminate redundancies in the explored part of the rule language by searching among *rule equivalence classes* where two rules are considered equivalent if they select the same data. The roots of this notion are in the formal concept analysis and knowledge discovery literature [2, 18] as a "condensed representation" of the information contained in a data collection. However, when used in conjunction with efficient traversal techniques [19], considering equivalence classes of rules, naturally also leads to substantial speed-ups in rule optimization similar to the gains from tight bounding functions [3, 4].

1.2 Contributions This paper builts on the abovementioned optimization techniques to devise an efficient optimal base learner for gradient boosting. The resulting rule learner outperforms greedy rule boosting as well as *RuleFit* in terms of the number of rules required for reaching a specific predictive performance. Thus, the presented approach is ideally suited to produce comprehensible rule ensembles. In detail:

1. We derive the per-stage rule objective function for second order gradient boosting and, as main theoretical result, show that the objective values of refinements of a selected query can be tightly upper-bounded in linear time in the number of selected data points (Thm 3.1, Sec. 3.1).

2. We integrate this bound into an anytime base learner for rule boosting based on branch-and-bound (Sec. 3.2). As innovation of independent interest, we give a formulation of rule equivalence class search that can be exploited in the OPUS propagation of pruning information (Thm. 3.2).

3. We empirically evaluate the resulting optimal rule boosting algorithm by comparing its predictive performance as well as its computational efficiency across 24 common benchmark classification and regression problems (Sec. 4).

We find that the optimal rule learner consistently outperforms the predictive performance of conventional greedy rule boosting as well as *RuleFit* when targeting comprehensible ensembles of 10 rules or less. Moreover, while its computational demands are naturally higher than that of myopic search, it does not require more than 2h (and is usually considerably faster) across all considered benchmarks—even using a standard PC and a preliminary Python implementation (`https://github.com/marioboley/realkd.py`).

2 Gradient Rule Boosting

2.1 Predictive Modeling Formally, we aim to model a **target variable** $Y \in \mathbb{Y}$ given some **feature vector** $X \in \mathbb{X}$ based on **training data** $\{(x_i, y_i)\}_{i=1}^n$ that has been sampled according to the joint distribution of X and Y. We focus on models in the form of a single-valued scoring function $f : \mathbb{X} \to \mathbb{R}$. For instance, in regression problems ($\mathbb{Y} = \mathbb{R}$), f typically models the conditional expectation of the target, i.e., $f(x) \approx E(Y \mid X = x)$, whereas in binary classification problems ($\mathbb{Y} = \{-1, 1\}$), f typically models the conditional **log odds**, i.e., $f(x) \approx \ln P(Y = 1 \mid X = x)/P(Y = -1 \mid X = x)$ and the conditional probabilities $p(y \mid x)$ are recovered by the sigmoid transform

$$p(y \mid x) = \sigma(f(x)) = (1 + \exp(-yf(x)))^{-1} \ .$$

Generally, the meaning of a score $f(x)$ is encapsulated in a positive **loss function** $l(y, f(x))$ that represents the cost of predicting $f(x)$ when the true target value is y. Specific examples are the **squared loss** $l(y, f(x)) = (y - f(x))^2$ for regression problems and the **logistic loss** $l(y, f(x)) = \log(1+\exp(-yf(x)))$ for classification problems. However, we only assume that l is twice differentiable and convex in its second argument. In general, our goal is to find a model f that minimizes the prediction **risk**, i.e., the expected loss according to the underlying distribution: $L(f) = E(l(y, f(x)))$. Since we have no information about that distribution other than the given training data, a learning algorithm can only compute the **empirical risk**:

$$\hat{L}(f) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, f(x_i)) \ .$$

However, naively optimizing the empirical risk might result in overfitting, i.e., a model with a poor actual prediction risk. To avoid this, learning algorithms typically optimize the **regularized (empirical) risk**

$$\hat{L}_\lambda(f) = \hat{L}(f) + \frac{\lambda}{n}\Omega(f) \tag{2.1}$$

where the empirical risk term is balanced with a model complexity term Ω weighted by a non-negative **regularization parameter** $\lambda \geq 0$. As we will see below, a positive regularization parameter $\lambda > 0$ is not only useful from a statistical perspective but also required by the gradient boosting framework when working with non-strictly convex loss functions (for which the second derivative can vanish).

2.2 Additive Rule Ensembles The basic functions considered in this paper are **rules** $r(x) = wq(x)$ where

$q : \mathbb{X} \to \{0, 1\}$ is a binary **query** function and $w \in \mathbb{R}$ is a **prediction weight**, which correspond to the rule's antecedent and consequent, respectively. The **extent** (coverage) of a rule/query are the indices of the instances selected by the query, i.e., $I(q) = \{i : q(x_i) = 1, 1 \leq i \leq n\}$. These functions represent IF-rules that predict the numeric value w for an input x if the condition represented by q holds for x and 0 otherwise (again, the interpretation of the predicted value depends on the modeling task).

Specifically, as possible antecedents, we consider the class of **conjunctive queries** that can be formed from a set of "atomic" **propositions** $\mathcal{P} \subseteq \{0, 1\}^{\mathbb{X}}$, i.e.,

$$\mathcal{Q} = \left\{ q \in \{0, 1\}^{\mathbb{X}} : q(x) = \prod_{p \in \mathcal{P}(q)} p(x), \;\; \mathcal{P}(q) \subseteq \mathcal{P} \right\}$$

where $\mathcal{P}(q)$ denotes the subset of propositions contained in query q. For instance, for the common case of $\mathbb{X} = \mathbb{R}^d$ one usually chooses threshold functions

$$\mathcal{P} = \{\mathbf{1}(x^{(j)} \leq x_i^{(j)}), \mathbf{1}(x^{(j)} > x_i^{(j)}) : \\ 1 \leq j \leq d, 1 \leq i \leq n\}$$

as propositions. With this choice, $\mathcal{Q}$ corresponds to the set of convex polytopes with axis-parallel sides. However, more general constructions are possible. In particular, it is easy to accommodate input spaces that mix continuous with categorical dimensions.

By forming sums of individual rules, we can then form the function class of **additive rule ensembles**

$$\mathcal{F} = \left\{ f \in \mathbb{R}^{\mathbb{X}} : f(x) = \sum_{i=1}^{k} w_i q_i(x), \;\; q_i \in \mathcal{Q}, w_i \in \mathbb{R} \right\}$$

where k is the **ensemble size**. Note that, assuming that $\mathcal{P}$ is closed under negation, tree models and by extension additive tree ensembles are defined using the same function class. However, each individual tree model has the restriction to be a sum of rules where the queries partition $\mathbb{X}$. This typically results in a much larger number of rules to accurately describe a given distribution [9].

RuleFit finds a function in $\mathcal{F}$ by restricting $\mathcal{Q}$ to a candidate query set of manageable size k' and then minimizing (2.1) over $\mathbb{R}^{k'}$ via convex optimization. To facilitate a small effective ensemble size k, the sparsity-inducing l1 complexity-measure $\Omega(f) = \sum_{i=1}^{k} |w_i|$ is used. In contrast, **rule boosting** grows an additive ensemble of arbitrary desired size by starting from the empty model $f_0 \equiv 0$ and then iteratively calling a **base learner** to find a new term $w_t q_t$ that minimizes the regularized loss when combined with the previously fixed part of the model $f_{t-1} = \sum_{i=1}^{t-1} w_i q_i$.

2.3 Objective Function Here, we adapt the "second order" gradient boosting approach popularized by *XGBoost* [5]: We define the model complexity as the squared Euclidean norm of the rules' consequents: $\Omega(f) = \sum_{i=1}^{k} w_i^2$. Then, to estimate the effect of r_t on (2.1) in a computationally convenient way, we approximate the regularized risk of f_t as

$$\begin{aligned} \hat{L}_\lambda(f_t) &= \frac{\lambda}{2n} \sum_{i=0}^{t} w_i^2 + \frac{1}{n} \sum_{i=1}^{n} l(y_i, f_{t-1}(x_i) + w_t q_t(x_i)) \\ &\simeq \hat{L}_\lambda(f_{t-1}) + \frac{w_t}{n} \left(\sum_{i \in I(q_t)} g_i \right) + \frac{w_t^2}{2n} \left(\lambda + \sum_{i \in I(q_t)} h_i \right) \end{aligned}$$

where we approximated the loss $l(y_i, f_{t-1}(x_i) + w_t q_t(x_i))$ by $l(y_i, f_{t-1}(x_i)) + g_i w_t q_t(x_i) + \frac{1}{2} h_i (w_t q_t(x_i))^2$ using the first and second order **gradient statistics**, g_i and h_i, of the prediction loss incurred at example i:

$$g_i = \frac{\mathrm{d} l(y_i, y)}{\mathrm{d} y}\Big|_{y = f_{t-1}(x_i)} \;, \quad h_i = \frac{\mathrm{d}^2 l(y_i, y)}{\mathrm{d} y^2}\Big|_{y = f_{t-1}(x_i)} \;.$$

The objective in the t-th iteration of gradient boosting is to minimize the regularized loss of f_t, or equivalently to maximize the **loss reduction**, $\hat{L}_\lambda(f_{t-1}) - \hat{L}_\lambda(f_t)$. For a fixed rule antecedent q_t, the associated optimal weight w_t is given by

$$w_t = -\frac{\sum_{i \in I(q_t)} g_i}{\lambda + \sum_{i \in I(q_t)} h_i} \tag{2.2}$$

Thus, we end up with the following objective function for q_t that we seek to maximize over the possible antecedents $q \in \mathcal{Q}$:

$$\mathrm{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i \right)^2}{2n \left(\lambda + \sum_{i \in I(q)} h_i \right)} \tag{2.3}$$

For instance, for regression problems using the squared loss we end up with the following gradient statistics and optimal rule weight:

$$\begin{aligned} g_i &= -2(y_i - f_{t-1}(x_i)) \\ h_i &= 2 \\ w_t &= \sum_{i \in I(q_t)} \frac{(y_i - f_{t-1}(x_i))}{\lambda/2 + |I(q_t)|} \;, \end{aligned}$$

and for classification with the logistic loss we have:

$$\begin{aligned} g_i &= -y_i \, p(-y_i \mid x_i) \\ h_i &= p(y_i \mid x_i) \, p(-y_i \mid x_i) \\ w_t &= \frac{\sum_{i \in I(q_t)} y_i \, p(-y_i \mid x_i)}{\lambda + \sum_{i \in I(q_t)} p(y_i \mid x_i) \, p(-y_i \mid x_i)} \;. \end{aligned}$$

For the latter, we can see that only for strictly positive λ, the optimal rule weight $w_t \to 0$ when $\prod_{i \in I(q_t)} p(y \mid x_i) \to 1$, i.e., when for all $i \in I(q_t)$ the modeled probabilities $p(y_i \mid x_i) \to 1$. In contrast, for $\lambda = 0$ we have $w_t \to 1$ even as the above conditional likelihood approaches 1. Thus, regularization seems very appropriate for this model (though even for $\lambda = 0$, $\mathrm{obj}(q_t) \to 0$ if the conditional likelihood approaches 1).

3 An Efficient Optimal Base Learner

In this section, we develop a practically efficient base learner for rule boosting that maximizes the objective function (2.3) exactly. As a convention, we fix some arbitrary order of the set of basic propositions $\mathcal{P} = \{p_1, \ldots, p_d\}$ and identify queries with ordered subsets of $\mathcal{P}$, i.e., $q = \{p_{i_1}, \ldots, p_{i_l}\}$ with $i_j < i_{j+1}$ for $1 \leq j < l$. The maximum index i such that $p_i \in q$ is called the **tail index**, denoted $\mathrm{tail}(q)$. In this framework, we say that a query q' is a **tail augmentation** of q, denoted as qp_i, if $q' = q \cup \{p_a\}$ for some $a > \mathrm{tail}(q)$. Finally, we call a query q a **prefix** of another query q', denoted by $q \sqsubseteq q'$, if q' can be generated from q via successive tail augmentations.

In a nutshell, the proposed base learner maximizes obj by enumerating candidate queries recursively via tail augmentations starting from the trivial query $q = \top$. Of course, naively this would always require a number of objective function evaluations exponentially in the number of propositions. We combat this exponential growth with two techniques: (i) search space pruning by bounding the objective value that can be attained by augmentations of a specific query and (ii) search space condensation by replacing the naive search space $\mathcal{Q}$ with a typically much smaller search space $\mathcal{Q}' \subseteq \mathcal{Q}$ in which the optimal objective value is still attained.

3.1 Search Space Pruning A general construction for effective bounding functions is often referred to as **tight optimistic estimator** in the rule discovery literature. Here, the term "tight" refers to the "selection-unaware" scenario where we relax the possible inputs to obj to include all possible index subsets; instead of just those that can be selected by a query q. This leads to the following relaxed function:

$$\begin{aligned} \mathrm{bnd}(q) &= \max\{\mathrm{obj}(J) : J \subseteq I(q)\} \\ &\leq \max\{\mathrm{obj}(q') : q' \supseteq q\} \ . \end{aligned}$$

While this formulation implies that bnd is an admissible pruning function for branch-and-bound search, in general it remains unclear how to compute it efficiently.

Fortunately, for our objective function (2.3) there is a general algorithm that computes $\mathrm{bnd}(I)$ in time $O(|I|)$ after an initial pre-sorting step of cost $O(n \log n)$ that has to be applied only once for each rule in the ensemble. This algorithm finds an optimal index subset $J^* \subseteq I$ by greedily selecting indices $i \in I$ in order of their corresponding loss derivative ratio g_i/h_i. The correctness of this approach is established with the following result.

THEOREM 3.1. *Let $i_1, \ldots i_m$ be the sequence of indices in the set $I \subseteq \{1, \ldots, n\}$ ordered such that*

$$\frac{g_{i_1}}{h_{i_1}} \geq \frac{g_{i_2}}{h_{i_2}} \geq \cdots \geq \frac{g_{i_m}}{h_{i_m}} \ .$$

Then there is an index subset J^* with $\mathrm{bnd}(I) = \mathrm{obj}(J^*) = \max\{\mathrm{obj}(J) : J \subseteq I\}$ that is given as a prefix or a suffix of the above sequence, i.e., for some $l > 0$

$$J^* = \{i_1, i_2, \ldots, i_l\} \quad \textit{and} \quad g_{i_l} > 0 \quad \textit{or} \tag{3.4}$$

$$J^* = \{i_{m-l+1}, \ldots, i_m\} \quad \textit{and} \quad g_{i_{m-l+1}} < 0 \ . \tag{3.5}$$

Proof. First, we can observe that for any optimal index subset J^* we have $J^* \subseteq I_+$ or $J^* \subseteq I_-$ with $I_+ = \{i \in I : g_i > 0\}$ and $I_- = \{i \in I : g_i < 0\}$: Indeed, for any $J \subseteq I$ with non-negative gradient sum $G_J = \sum_{i \in J} g_i \geq 0$ and $j \in J \cap I_-$ we have $G_{J \setminus \{j\}} > G_J$. Moreover, due to the positivity of h_j, we have $\sum_{i \in J} h_i = H_J > H_{J \setminus \{j\}}$, and thus $\mathrm{obj}(J \setminus \{j\}) > \mathrm{obj}(J)$. The same is true for J with $G_J \leq 0$ and $j \in J \cap I_+$. Thus, it is sufficient to show (3.4) for the case $J^* \subseteq I_+$ and (3.5) for $J^* \subseteq I_-$. Moreover, by symmetry of the objective function it suffices to show the first case (as $\{i_{m-l+1}, \ldots, i_m\} \subseteq I_-$ is optimal wrt g and h iff $\{i_1, \ldots, i_l\} \subseteq I_+$ is optimal wrt $-g$ and h).

For an index subset $J \subseteq I_+$, let $\mathrm{trans}(J)$ be the number of transpositions (or sorting violations) of J, i.e., the number of ordered index pairs i, j such that $j \in J$ and $i \in I_+ \setminus J$ but $g_i/h_i \geq g_j/h_j$. We will show that for each $J \subseteq I_+$ with $\mathrm{trans}(J) > 0$ there is an index set $J' \subseteq I_+$ with $\mathrm{trans}(J') < \mathrm{trans}(J)$ such that $\mathrm{obj}(J') \geq \mathrm{obj}(J)$. Consequently, there is an optimal index set J^* with $\mathrm{trans}(J^*) = 0$ as required.

Let $\mathrm{trans}(J) > 0$ and $j \in J$ and $i \in I_+ \setminus J$ an index pair with $g_i/h_i \geq g_j/h_j$ (or equivalently $g_i h_j \geq g_j h_i$). In the special case that $J = \{j\}$ and $\lambda = 0$, we can directly see that, up to a factor $2n$,

$$\begin{aligned} \mathrm{obj}(J \cup \{i\}) - \mathrm{obj}(J) &= \frac{(g_i + g_j)^2}{h_i + h_j} - \frac{g_j^2}{h_j} \\ &= \frac{g_i^2 h_j + g_j(2 g_i h_j - g_j h_i)}{(h_i + h_j) h_j} \geq 0 \ . \end{aligned}$$

For the case that $\lambda > 0$ or $|J| > 1$, we can apply Lemma A.1 from the supplementary material, which

sort data s.t. $g_{i-1}/h_{i-1} > g_i/h_i$, $1 < i \leq n$
init boundary **to** empty priority queue
$q^* = \top$, $I = \{1, \ldots, n\}$
$A = \{(i, \infty, i) : 1 \leq i \leq d\}$ *# (aug idx, bound, crit idx)*
push (q^*, I, A) **to** boundary
while boundary **not** empty
  pop (q, I, A) **from** boundary
  init A' **to** empty set
  for $(a, b, c) \in A$ **if** $q \not\to p_a, \mathrm{obj}(q^*) < b, \mathrm{tail}(q) \leq c$
    $I_a = I \cap I(p_a)$ *# via merge retaining order*
    $q^* = \arg\max\{\mathrm{obj}(qp_a), \mathrm{obj}(q^*)\}$
    $b' = \mathrm{bnd}(I_a)$ *# via Thm.* 3.1
    $c' = \begin{cases} c & , \text{ if } c < a \text{ and } q \not\to p_c \\ \mathrm{crt}(qp_a) & , \text{ otherwise} \end{cases}$ *# (3.7)*
    add (a, b', c') **to** A'
  for $(a, b, c) \in A'$ **if** $a = c$
    push $(qp_a,\ I_a,\ A')$ **to** boundary
$q_{\mathrm{short}} =$ apx. shortest equivalent to q^*
return $r(x) = wq_{\mathrm{short}}(\mathrm{x})$ *# w via Eq.* (2.2)

Algorithm 1: *Branch-and-Bound Base Learner.*

states that for positive numbers r, a, c and strictly positive numbers s, b, d with $a/b \geq c/d$,

$$\frac{(r+c)^2}{s+d} \leq \max\left\{\frac{r^2}{s}, \frac{(r+a+c)^2}{s+b+d}\right\} \ .$$

By setting $r = \sum_{t \in J \setminus \{j\}} g_t$, $s = \lambda + \sum_{t \in J \setminus \{j\}} h_t$, $a = g_i$, $b = h_i$, $c = g_j$, and $d = h_j$ it follows that

$$\mathrm{obj}(J) \leq \max\{\mathrm{obj}(J \setminus \{j\}), \mathrm{obj}(J \cup \{i\})\} \ .$$

Since $\mathrm{trans}(J \setminus \{j\}) < \mathrm{trans}(J)$ as well as $\mathrm{trans}(J \cup \{i\}) < \mathrm{trans}(J)$, $J \setminus \{j\}$ or $J \cup \{i\}$ is an index set with strictly fewer transpositions that dominates J. ∎

3.2 Search Space Condensation An important observation about the objective function $\mathrm{obj}(q)$ is that it is a function of the query extent $I(q)$. That is, queries q, q' with equal extents $I(q) = I(q')$ have the same objective value and can thus be considered (empirically) **equivalent**, denoted $q \leftrightarrow q'$, for the purpose of optimization. Similarly, let us say that a query q (empirically) **implies** another query q', denoted $q \to q'$, if $I(q') \supseteq I(q)$.

The following recursive construction of Uno et al. [19] allows for an efficient enumeration of a single **core query** per equivalence class: The trivial query $q = \top$ is a core query. Moreover, a tail augmentation qp_i of a core query q is also a core query if $q \not\to p_i$ and

$$\text{for all } j < i, \text{ if } q' \to p_j \text{ then } q \to p_j \ . \tag{3.6}$$

It is straightforward to show that the core queries form a prefix tree rooted in $\top$. This allows to enumerate all core queries in a tree traversal where all children of a given tree node q can be found by checking all tail augmentations qp_i and filter for those that satisfy the prefix preservation condition (3.6).

Importantly, the set of core queries is determined by an arbitrary order imposed on the set of propositions $\mathcal{P}$. Also, while core queries are inclusion-minimal in their equivalence class, there can be shorter queries describing a specific extension. Thus, to encode a generality bias that is independent of the proposition order, it makes sense to convert an optimized core query to an (approximately) shortest equivalent query in a post-processing step (using a greedy algorithm [see 3]).

While the core query approach leads to a substantial reduction of the search space, the prefix preservation checks have a notable cost (worst case $\Omega(d\,|I(q)|)$). The following novel notion us to do that: for core query q and tail extension p_i define the **critical index** of qp_i as

$$\mathrm{crt}(qp_i) = \min\{j : qp_i \to p_j, q \not\to p_j\} \leq i \ .$$

The critical index of a tail augmentation is naturally determined when checking the prefix-preservation condition (3.6), which holds if and only if $\mathrm{crt}(qp_i) = i$. Knowing the exact critical index can be used to reduce the number of required checks in the successors of certain sibling nodes as shown in the following theorem.

THEOREM 3.2. *Let q and q' be core queries such that $qp_i \sqsubseteq q'$ and $c = \mathrm{crt}(qp_j) < j$ for $j > i$. Then*

$$\textit{if } q' \not\to p_c \textit{ then } q'p_j \textit{ is not a core query} \tag{3.7}$$
$$\textit{if } c < i \textit{ then } q'p_j \textit{ is not a core query} \ . \tag{3.8}$$

Proof. For property (3.7) we can observe that $q'p_j \to p_c$ because $I(p_c) \supseteq I(qp_j) \supseteq I(q'p_j)$. Together with the premise that $q' \not\to p_c$ it follows that $q'p_c$ does not satisfy the prefix condition (3.6). Property (3.8) follows as a special case of (3.7) by showing that $c < i$ implies that $q' \not\to p_c$. Indeed, assuming that $q' \to p_c$ there needs to be a prefix-minimal query q'' with $qp_i \neq q'' \sqsubseteq q'$ such that $q'' \to p_c$. However, then q'' is not a core query and neither are its suffixes including q', which contradicts our assumptions. ∎

Algorithm 1 shows how Thms. 3.1 and 3.2 are integrated into the OPUS/branch-and-bound framework. As usual, every query q that still needs to be considered for augmentation is kept in a priority queue (boundary) along with a list of augmentation elements (A) that are promising for that query. As novel extension, this list does not only contain the augmentation elements (a) and bounds (b) to their objective value but also their

previously determined critical index (c). This information is used in line 9 to prune augmentations if they lead to (i) equivalent queries ($q \to p_a$), (ii) to queries of insufficient objective value ($\text{obj}(q^*) \geq b$), or (iii) to non-core queries ($\text{tail}(q) > c$). The latter condition is correct due to (3.8), which allows us to recursively prune all refinements $q' \sqsupseteq qp_a$. Additionally, property (3.7) allows to skip the prefix preservation check for qp_a itself, as it is guaranteed to lead to a non-core query, although a can still be a valid augmentation index for extensions of qp_a. Finally, in line 18 the result query is converted to a shortest equivalent query.

We close the description of the algorithm with noting an important property that helps coping with hard inputs: It is an anytime algorithm in the sense that we can obtain the current q^* as an approximation to the optimal query at any time when terminating early. Importantly, we also obtain the **multiplicative approximation guarantee** $\text{obj}(q^*)/\max\{b : (a, b, c) \in \text{boundary}\}$ for this current guess. In fact, when we are a priori satisfied with an α-approximation to the optimal query, we can directly relax the corresponding condition in line 9 for even more effective pruning.

4 Evaluation

In this section we present comparative results of optimal and greedy rule boosting as well as *RuleFit* when fitting small rule ensembles to a range of synthetic and real-world prediction problems. Here, *greedy rule boosting* refers to using the standard rule learner that, starting from the empty conjunction, chooses one proposition at a time maximizing the increase in (2.3) and stops if no improvement can be achieved. In addition to the predictive performance we also assess the learners' computational efficiency. To that end, we use a publicly available preliminary Python implementation of the boosting algorithms and the implementation of *RuleFit* available on PyPi, which is based on the highly efficient *XGBoost* for obtaining the input forest (see Supplementary Materials for additional details, links, and results).

Setting Based on our goal of producing comprehensible models, we investigate the rule learner's performance for small ensemble sizes of up to ten rules. While the boosting algorithms can naturally produce any desired ensemble size, for *RuleFit* this entails tuning the l1-regularization parameter to realize as many small ensemble sizes as possible (and average the performance for specific sizes). Since this approach does not necessarily yield all ensemble sizes from 1 to 10, we interpolate performances for missing sizes by considering the area under the size/performance curve (see Fig. 2).

Specifically, for regression problems we measure the models' R^2 score, and for classification problems we measure the area under the ROC curve. The required computation time is assessed for the greatest ensemble size (10). For all prediction problems, the measured performance is averaged over five repetitions with different training/test splits. The selected prediction problems contain all problems included in the `scikit-learn.datasets` module and, motivated by their interpretability, the most popular problems from Kaggle competitions.

Overall Results The overall results are summarized in Tab. 1 along with the basic problem characteristics. In terms of predictive performance, we find that optimal rule boosting consistently outperforms greedy rule boosting, which in turn outperforms the *RuleFit* models, for all 24 problems (*rendering the hypothesis that $S_{opt} > S_{grd} > S_{rf}$ significant at the 0.01-level based on two one-sided sign-tests; where S is the average performance over 1-10 rules of the method on a random dataset*). While for five problems, optimal and greedy boosting are essentially en par, for 19 problems the average performance difference is more than 0.01, and for four out of those the difference is more than 0.03 score points. Importantly, these results reflect average differences across ensembles sizes. For individual sizes the advantage can be much more pronounced (see below).

In terms of computation time we can observe that, while greedy boosting generally outperforms optimal boosting, the required computation times are in the same order of magnitude (less than a factor 5 apart) for 13 of the problems. For eight further problems, greedy is one order of magnitude faster than optimal (factor between 5 and 50), and only for three problems (*IBM-HR*, *boston*, *mobile prices*) optimal rule boosting requires two orders of magnitude more computation time. Importantly, for all but one problem, an optimal rule boosting ensemble is found in less than one hour of computation time on a personal computer without a highly optimized implementation.

Analysis When analyzing the predictive figures in more detail, we find that optimal and greedy ensembles generally have a lot of rules in common. However, an individual sub-optimal rule in an ensemble can be enough to set greedy ensembles behind. For instance for the *used cars* price regression problem, the 5-rule boosting ensembles for the two base learners are:

```
optimal base learner
1: +16192 if PS>=100 & year>=2003
2: +10596 if count<=86 & PS>=180 & year>=2009
3: - 8360 if PS in [100,180] & year in [2003,2009]
4: + 5837 if PS>=180 & year<=2003
5: + 2497 if km<=70000

greedy base learner
1: +16202 if PS>=100 & year>=2003
```

dataset	feat.	rows	#rules/score AUC			comp. time (s)			unbnd. rf	
			rf	grd	opt	rf	grd	opt	score	#rul.
breast cancer	30	569	0.908	0.949	**0.953**	3.0	13.9	44.9	0.984	46
digits (5)	64	3915	0.500	0.923	**0.927**	10.0	266.4	947.7	0.976	228
gender recog	20	3168	0.940	0.942	**0.956**	7.9	77.0	1642.9	0.983	113
IBM HR	32	1470	0.500	0.677	**0.679**	4.1	44.1	5925.0	0.674	173
iris (1)	4	150	0.689	0.910	**0.915**	1.3	2.5	10.1	0.944	99
wine (1)	13	178	0.794	0.931	**0.942**	1.5	3.2	4.0	0.978	85
classification2	8	2000	0.753	0.869	**0.887**	5.3	27.2	228.2	0.906	155
telco churn	18	7043	0.500	0.776	**0.779**	16.3	82.6	517.3	0.725	88
tic-tac-toe	27	958	0.699	0.752	**0.804**	3.1	14.4	15.6	0.990	194
titanic	7	1043	0.699	0.857	**0.859**	3.0	14.7	144.1	0.837	35
boston	13	506	0.163	0.544	**0.565**	2.4	10.1	585.1	0.877	151
demographics	13	6876	0.209	0.335	**0.343**	14.5	112.3	941.6	0.529	170
insurance	6	1338	0.225	0.747	**0.751**	3.6	12.0	16.2	0.835	236
load diabetes	10	442	0.182	0.295	**0.308**	2.1	5.7	51.9	0.456	53
friedman1	10	2000	0.209	0.508	**0.553**	5.9	24.9	51.4	0.977	231
friedman2	4	10000	0.265	0.788	**0.808**	5.8	16.9	18.5	0.999	638
friedman3	4	5000	0.161	0.508	**0.524**	4.8	16.9	22.5	0.881	192
mobile prices	20	2000	0.132	0.770	**0.779**	7.2	39.5	2437.3	0.934	105
red wine quality	11	1599	0.109	0.256	**0.267**	4.1	22.0	623.1	0.449	138
suicide rates	5	27820	0.206	0.298	**0.302**	45.3	243.1	259.8	0.396	249
used cars	4	1770	0.269	0.689	**0.720**	4.2	12.5	13.3	0.962	283
videogamesales	6	16327	0.116	0.840	**0.860**	34.7	131.0	148.8	0.999	991
life expectancy	21	1649	0.246	0.582	**0.599**	6.4	68.4	161.2	0.955	245
world happiness	8	315	0.274	0.599	**0.653**	1.8	5.2	34.3	0.969	108

Table 1: Empirical results overview. Table shows area under #rules/score curve and computation time for *RuleFit* (rf), greedy rule boosting (grd), and optimal rule boosting (opt). As reference, also prediction score and number of rules for *RuleFit* with unbounded number of rules are given. Score is ROCAUC for classification (upper section) or R^2 for regression (lower section). All results are mean values over five different random 80/20 train/test splits.

```
2: +10612 if count<=86 & PS>=180 & year>=2009
3: + 9791 if year>=2015
4: + 5790 if PS>=180 & year<=2003
5: - 8414 if PS in [100,180] & year in [2003,2009]
```

That is, both base learners yield the same two initial rules. At position 3, however, the complex interaction between engine power and construction year is not (yet) discovered by the greedy base learner. Instead, it is replaced by a single condition rule with much smaller predictive gain, which accounts for a substantial performance difference for ensemble sizes 3 and 4; until the rule is finally discovered at position 5 (see Fig. 2a).

Another example is the *tic-tac-toe* classification problem, where for size 5 we end up with the ensembles:

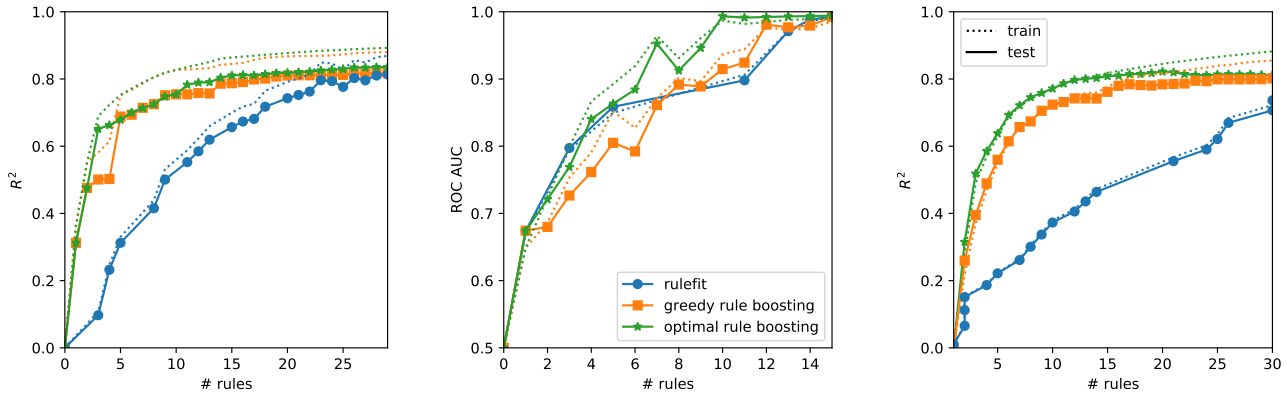
```
optimal base learner
1: +1.187 if mid-mid==x
2: +1.717 if top-lft==x & top-mid==x & top-rgt==x
3: +1.721 if bot-lft==x & bot-mid==x & bot-rgt==x
4: +1.671 if bot-rgt==x & mid-rgt==x & top-rgt==x
5: -2.344 if top-lft==o & top-mid==o & top-rgt==o

greedy base learner
1: +1.187 if mid-mid==x
2: +1.211 if bot-rgt==x & mid-mid==b
3: -2.671 if top-lft==o & top-mid==o & top-rgt==o
4: +1.487 if bot-lft==x & bot-mid==x & bot-rgt==x
5: +1.685 if top-lft==x & top-mid==x & top-rgt==x
```

That is, after the initial "statistical" rule, optimal rule boosting proceeds with adding exact "symbolic" win conditions to its ensembles. Greedy rule boosting creates a very similar ensemble but inserts a largely ineffective statistical rule at position 2, which again leads to a size/performance disadvantage (see Fig. 2b).

5 Conclusions

Our analysis of the second order gradient boosting objective yields a computationally efficient optimal base learner for rule boosting. As our results indicate, this base learner consistently outperforms a conventional greedy rule learner in terms of its predictive performance. Besides the previous lack of an efficient search algorithm, the advantage of an optimal base learner might have been overlooked thus far, because it tends



(a) cars (b) tic-tac-toe (c) friedman1

Figure 2: Predictive performance against number of rules for selected prediction problems.

to vanish for large ensemble sizes. However, as demonstrated by the noisy parity example of Fig. 1, some problems with higher-order interactions are learned accurately by optimal rule boosting with relatively few rules but are intractable by greedy rule boosting even for very large ensemble sizes.

Thus, the presented approach adds a tool of general interest to the boosting framework that should in particular be considered when seeking comprehensible, i.e., small, models: Not only are the predictive gains most pronounced for a small number of rules. In this regime, also the required computational overhead can be tolerated more easily. One direction to push the comprehensibility/accuracy trade-off even further, is to completely replace the stage-wise fitting paradigm and consider "global optimization" approaches. An efficient algorithm for this has been proposed for rule *lists* and the specific case of accuracy-based classification [1]. Lifting these ideas to additive rule ensembles, ideally of the same universality as the gradient boosting framework, is a natural next goal.

References

[1] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin. Learning certifiably optimal rule lists for categorical data. *JMLR*, 18(1):8753–8830, 2017.

[2] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Computational Logic*, pages 972–986. Springer, 2000.

[3] M. Boley and H. Grosskreutz. Non-redundant subgroup discovery using a closure system. In *ECMLPKDD*, pages 179–194. Springer, 2009.

[4] M. Boley, B. R. Goldsmith, L. M. Ghiringhelli, and J. Vreeken. Identifying consistent statements about numerical data with dispersion-corrected subgroup discovery. *Data Min Knowl Discov*, 31(5):1391–1418, 2017.

[5] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794, 2016.

[6] W. W. Cohen and Y. Singer. A simple, fast, and effective rule learner. *AAAI/IAAI*, 99(335-342):3, 1999.

[7] K. Dembczyński, W. Kotłowski, and R. Słowiński. Maximum likelihood rule ensembles. In *ICML*, pages 224–231, 2008.

[8] K. Dembczyński, W. Kotłowski, and R. Słowiński. Ender: a statistical framework for boosting decision rules. *Data Min Knowl Discov*, 21(1):52–90, 2010.

[9] X. Fan, B. Li, and S. Sisson. Rectangular bounding process. In *Advances in Neural Information Processing Systems*, pages 7620–7630, 2018.

[10] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Ann Stat*, pages 1189–1232, 2001.

[11] J. H. Friedman, B. E. Popescu, et al. Predictive learning via rule ensembles. *Ann Appl Stat*, 2(3):916–954, 2008.

[12] J. Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

[13] J. Fürnkranz and A. Knobbe. Guest editorial: Global modeling using local patterns. *Data Min Knowl Discov*, 21 (1):1–8, 2010.

[14] H. Grosskreutz, S. Rüping, and S. Wrobel. Tight optimistic estimates for fast subgroup discovery. In *ECMLPKDD*, pages 440–456. Springer, 2008.

[15] H. Lakkaraju, S. H. Bach, and J. Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *KDD*, pages 1675–1684, 2016.

[16] F. Lemmerich, M. Atzmueller, and F. Puppe. Fast exhaustive subgroup discovery with numerical target concepts. *Data Min Knowl Discov*, 30(3):711–762, 2016.

[17] S. Morishita and J. Sese. Transversing itemset lattices with statistical metric pruning. In *SIGMOD-SIGACT-SIGART*, pages 226–236. ACM, 2000.

[18] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing iceberg concept lattices with titanic. *Data Knowl Eng*, 42(2):189–222, 2002.

[19] T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Disc Science*, pages 16–31. Springer, 2004.

[20] G. I. Webb. Opus: An efficient admissible algorithm for unordered search. *AI Research*, 3:431–465, 1995.

[21] G. I. Webb. Discovering associations with numeric variables. In *KDD*, pages 383–388. ACM, 2001.

[22] G. Zhang and A. Gionis. Diverse rule sets. In *KDD*, pages 1532–1541, 2020.

A Technical Results

LEMMA A.1. *Let r, a, and c be positive real numbers, and s, b, and d be strictly positive numbers such that $a/b > c/d$. It holds that*

$$\frac{(r+c)^2}{s+d} \leq \max\left\{\frac{r^2}{s}, \frac{(r+a+c)^2}{s+b+d}\right\} \quad . \tag{A.1}$$

Proof. Setting $x = a + c$, and $y = b + d$, we define for $\alpha \in [0, 1]$ the function

$$z(\alpha) = \frac{(r+\alpha x)^2}{s+\alpha y} \quad .$$

See Fig. 3 for an illustration. With this definition we have $z(0) = r^2/s$, $z(1) = (r + a + c)^2/(s + b + d)$ and

$$\begin{aligned} z\left(\frac{c}{a+c}\right) &= \frac{(r+c)^2}{s+c(b+d)/(a+c)} \\ &\geq \frac{(r+c)^2}{s+d} \quad , \end{aligned}$$

where the inequality holds because of our premise that $a/b \geq c/d$, or equivalently that $b/a \leq d/c$, which implies that

$$s + d = s + c\frac{d}{c} \geq s + c\frac{b+d}{a+c} \quad .$$

Furthermore, $z(\alpha)$ is convex, as we can see by finding its second derivative

$$z''(\alpha) = \frac{2(ry - sx)^2}{(s+\alpha y)^3} \quad ,$$

which is positive for $\alpha \in [0, 1]$ based on the positivity of s, b, and d. The convexity of z implies that its maximum is attained at the boundary, i.e., either for $\alpha = 0$ or $\alpha = 1$. Thus,

$$\begin{aligned} \frac{(r+c)^2}{s+d} &\leq z\left(c/(a+c)\right) \\ &\leq \max\{z(0), z(1)\} \\ &= \max\{r^2/s, (r+a+c)^2/(s+b+d)\} \end{aligned}$$

as required. □

B Details of Experimental Setup

The code of all experiments as well as exact versions of datasets and software dependencies can be retrieved from `https://github.com/SimonTeshuva/interpretable_Rule_Ensemble`.

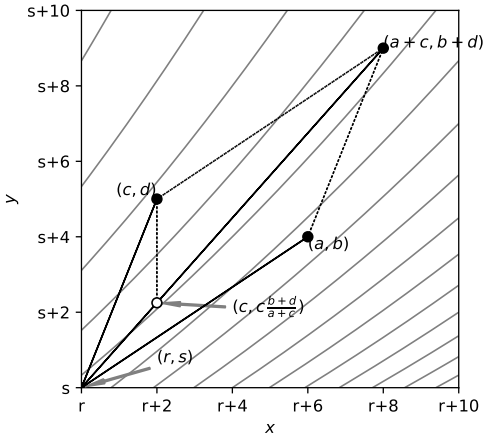


Figure 3: Illustration of the proof of Lemma A.1.

Dataset selection The empirical study targeted interpretable datasets of small to medium size. For that, all datasets from `sklearn.datasets` have been included that are loaded by a function with name starting `load_`. That is, the large datasets that are "fetched" online have been excluded. Additionally, all Kaggle datasets with more than 300 votes (as of December 2019) have been retrieved filtered by the following criteria:

1. Competition has a clearly defined regression or classification problem given in a csv file.
2. File is between 50KB and 1MB.
3. For similar datasets only the higher ranked one was selected.

Rule Learner Configuration For both greedy and optimal rule boosting, the regularization parameter of the base learner was optimized from the following options:

$$\lambda \in \{0.0001, 0.001, 0.01, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50\} \quad ,$$

where for the optimal base learner some small options were skipped whenever the required computation time exceeded 2h (see additional experiments below for a discussion of the effect of λ on the computation time). For *RuleFit*, first boundary regularization values were found that create 1 and approximately 11 rules, respectively. Then, the algorithm was run with 50 regularization values that linearly interpolate between these boundary values, resulting in the reported average performance.

Computing Environment All experiments were carried out on a personal laptop computer equipped with an Intel i5-6200U (4) @ 2.800GHz CPU, an integrated Intel Skylake GT2 GPU and 8GB RAM. The

system was running Ubuntu 18.04.4 LTS x86_64 and Python: 3.6.9 .

C Additional Experiments

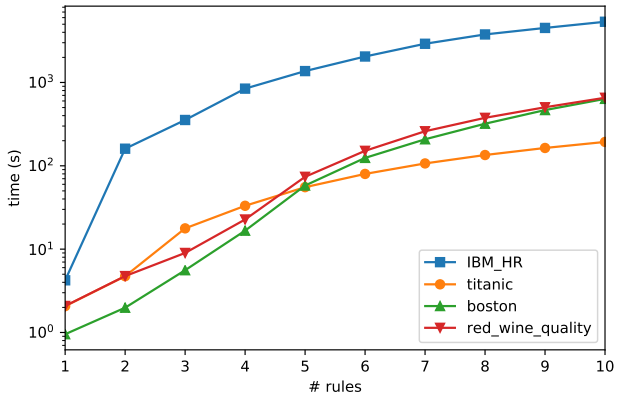


Figure 4: Computation time of optimal rule boosting versus number of rules.

The experimental results presented in the main text are mainly focused on predictive performance and only include summary computation times for the completely solving the considered prediction tasks. Beyond this, it is of practical interest to further assess the factors with which one can influence the computation time of optimal rule boosting.

Scaling in number of rules A first question is how the computation time scales with the number of rules. Naively one could assume a simple linear dependence. However, we typically find a super-linear behavior where the required computation time increases very fast for the first couple of rules and then starts to level off for larger ensemble sizes (see Fig. 4). This phenomenon can be explained by the coverage of the discovered rules: In the first couple of iterations there are typically still strong general rules that are identified early in the search process such that a lot of low coverage rules do not have to be explored due to the optimistic estimator pruning. Once these patterns are exhausted, the search has to explore more specific rules, which suffers from the corresponding combinatorial growth of the relevant part of the search space.

Scaling in regularization parameter This observation suggest to use the regularization parameter λ as a principled means to reduce the required computation time (in addition to relaxing the exact optimization to an approximation with multiplicative guarantee, as briefly discussed in the main text). Analyzing the objective function, we find that the gain for covering additional data point (with the same sign as the overall gradient sum) increases with λ. Consequently, greater λ values result in optimal rules with greater coverage, which again allows additional pruning and, hence, lower computation time. Fig. 5 shows this effect for the same datasets as considered above.

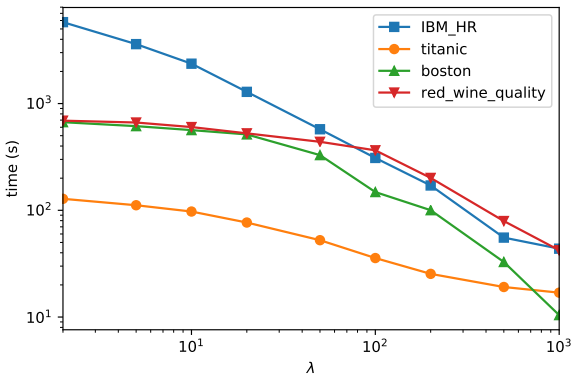


Figure 5: Computation time of optimal rule boosting (ensemble size 10) versus regularization parameter λ.

Effect of pruning techniques While the above two insights are perhaps the most important from a user perspective, for guiding further algorithmic development it is also instructive to investigate the relative importance of the different pruning mechanism employed in Algorithm 1. We can distinguish between the two mechanisms of bounding the objective function (enabled by Thm. 3.1) and avoiding the redundant enumeration of equivalent queries (through the core query construct of LCM). Moreover, it is instructive to differentiate the immediate application of these mechanism as in standard branch-and-bound and the propagated application in OPUS, which in the case of equivalence pruning is enabled by Thm. 3.2.

Tab. 2 contains the number of activations of each of these mechanisms for eight exemplary datasets. We can generally confirm previous studies that found that typically bounding and non-redundancy contribute to pruning in the same order of magnitude (although variations are possible where either mechanism is much more important than the other). As an interesting novel observation, we can see that propagating equivalence-related pruning information through the critical index construction can be equally effective as propagation of the objective bound (note that the table gives a conservative estimate, as additional non-recursive pruning enabled by (3.8) is not included in these numbers). This observation underlines the contribution of Thm. 3.2 to enable an overall efficient rule optimization.

dataset	immediate		propagated	
	bnd	equiv	bnd	equiv
boston	1280132	941252	146216	405358
breast	151420	88881	18902	54427
friedman1	227373	4473	2981	0
iris	15881	26208	2255	3310
red wine qu.	2881173	149112	54389	114079
tic-tac-toe	4303	7772	2179	0
titanic	106374	911809	1969	27241
used cars	1347	990	183	1
wrl. happi.	148327	17080	26420	5904

Table 2: Number of pruning rule activations: bounding the objective function (bnd) and avoiding redundant generation of equivalent queries (equiv), separated by immediate application and through OPUS propagation of pruning information (propagated).