MOSS: Multi-Objective Optimization for Stable Rule Sets

Citation: Brian Liu and Rahul Mazumder. 2025. MOSS: Multi-Objective Optimization for Stable Rule Sets. In Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25). Association for Computing Machinery, New York, NY, USA, 1753–1764.

Published Version: https://doi.org/10.1145/3711896.3737055

Publisher: ACM|Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2

Permanent Link: https://hdl.handle.net/1721.1/162624

Version: Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

MOSS: Multi-Objective Optimization for Stable Rule Sets

Brian Liu
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
briliu@mit.edu

Rahul Mazumder
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
rahulmaz@mit.edu

Abstract

We present MOSS, a multi-objective optimization framework for constructing stable sets of decision rules. MOSS incorporates three important criteria for interpretability: sparsity, accuracy, and stability, into a single multi-objective optimization framework. Importantly, MOSS allows a practitioner to rapidly evaluate the trade-off between accuracy and stability in sparse rule sets in order to select an appropriate model. We develop a specialized cutting plane algorithm in our framework to rapidly compute the Pareto frontier between these two objectives, and our algorithm scales to problem instances beyond the capabilities of commercial optimization solvers. Our experiments show that MOSS outperforms state-of-the-art rule ensembles in terms of both predictive performance and stability.

CCS Concepts

• **Computing methodologies** → **Machine learning**.

Keywords

Machine Learning, Optimization, Interpretability, Stability

ACM Reference Format:
Brian Liu and Rahul Mazumder. 2025. MOSS: Multi-Objective Optimization for Stable Rule Sets. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 29 pages. https://doi.org/10.1145/3711896.3737055

1 Introduction

Rule sets are prized in high-stakes applications of machine learning, such as criminal justice and healthcare operations, for balancing a high degree of transparency with good predictive performance [17, 28]. These ensembles consist of decision rules, or sequences of if-then antecedents (i.e. splits) that partition a datasets and assign a prediction to the partition; summing these predictions produces the output of the ensemble [14]. Sparse rule ensembles, of a manageable set of less than 20 or so decision rules [21], are especially useful in high-stakes applications since these rules can be audited by hand for fairness or bias concerns. However, sparse rule sets should also be stable and accurate to be considered trustworthy [34]. Intuitively, an algorithm that produces vastly different rule sets across small data perturbations is unlikely to be trusted. Likewise, rule sets that have insufficient explanatory power, or that perform poorly out-of-sample, should not be relied upon[1].

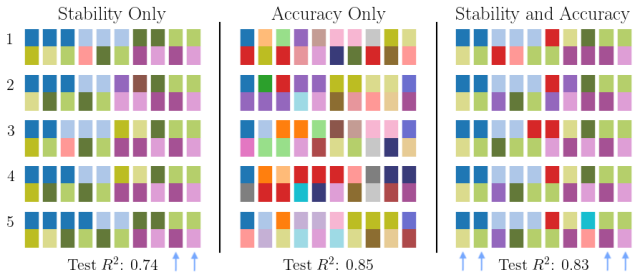

Figure 1: Rule sets constructed across 5 train-test splits. The blue arrows show rules that are stable across all splits.

Sparsity, accuracy, and stability are three competing objectives to consider when constructing interpretable models. The trade-off between model sparsity and accuracy has been well-explored, and popular frameworks such as GLMNET and L0LEARN allow practitioners to rapidly evaluate how model size impacts predictive performance [18, 19]. The trade-off between model stability and accuracy, on the other hand, is poorly understood. Frameworks that improve model stability, such as stability selection [24], do not account for predictive accuracy when constructing models. We illustrate this in the motivating example below, where we use a stability selection-based framework to construct stable rule sets [3, 24].

The general stability selection framework for constructing rule sets proceeds as follows. Given a dataset, we apply a sampling procedure such as the bootstrap to generate multiple perturbed datasets. We construct a set of decision rules on each perturbation; one such algorithm, SIRUS, does so byfi tting a single decision tree and taking the leaf nodes as decision rules [3]. Across all perturbations, we compute the frequency at which unique decision rules appear, and we return the set of rules with frequencies above some threshold. By extracting rules that are consistently constructed across data perturbations, stability selection improves model stability. However, the framework does not select rules with respect to their accuracy on the original data. As such, the predictive performance of the rule sets may suffer.

In the left panel in Figure 1, we apply stability selection (SIRUS) to construct 9 decision rules of interaction depth 2, across 5 different train-test partitions of the GALAXY regression dataset [30]. Each rectangle in the panel represents a decision rule and contains two vertically-stacked squares to represent each split. The squares are

[1]We direct readers to the Predictability, Computability, and Stability framework for veridical data science[25, 34, 35] for an in-depth discussion on criteria for trustworthiness in interpretable machine learning.

KDD '25, Toronto, ON, Canada

ACM ISBN 979-8-4007-1454-2/2025/08
https://doi.org/10.1145/3711896.3737055

color-coded with respect to split locations, i.e, squares with the same color are identical splits with respect to features and values. We observe that two rules are consistently recovered across partitions (indicated by the blue arrows) and that the average out-of-sample R^2 of the constructed rule set is **0.74**. In the center panel in Figure 1, we use RuleFit [14] to construct 9 decision rules with respect to only predictive accuracy. We observe that the constructed rules perform much better, with an average out-of-sample R^2 of **0.85**, however, the rules are unstable. No rules are consistently constructed across all train-test splits. From this example, we see that there is a trade-off between stability and accuracy that should be explored when constructing sparse rule sets.

In this paper we introduce MOSS, a multi-objective optimization framework for constructing stable rule sets. In contrast to existing algorithms, such as SIRUS and RuleFit, MOSS incorporates stability, sparsity, and accuracy into a single unified optimization problem. Importantly, we develop a novel optimization formulation and algorithm in MOSS that allows our framework to efficiently explore the Pareto frontier between accuracy and stability. As such, practitioners can use MOSS to rapidly evaluate the trade-off between these two objectives in order to select an appropriate model. Our experiments show that MOSS can extract sparse stable models that outperform state-of-the-art rule-based algorithms. In fact, when we apply MOSS to our motivating example discussed above we construct rule sets more stable than SIRUS with better out-of-sample predictive performance. In the right panel in Figure 1 we show the rule sets constructed by MOSS. We observe that 4 rules are consistently constructed across all train-test splits and that the rule sets have an average out-of-sample R^2 of **0.83**. We summarize the contributions of our paper below.

- We develop a multi-objective optimization framework for constructing rule sets that jointly accounts for accuracy, stability, and sparsity (§3.1).
- We develop specialized optimization algorithms (§3.2) that allow practitioners to efficiently explore the Pareto frontier between model accuracy and stability (§3.3).
- We show that MOSS can outperform state-of-the-art rule-based algorithms in terms of both stability and accuracy (§4.2).

The remainder of our paper is organized as follows. Wefirst discuss some preliminaries on model stability and the stability selection framework. We show that stability selection can be re-interpreted from an optimization viewpoint, which motivates our exploration into multi-objective optimization for both stability and accuracy. We then present the MOSS framework, along with the specialized optimization algorithm we use to solve problems in MOSS. We conclude with our experimental results, where we evaluate the stability and predictive performance of rule sets created using MOSS against various state-of-the-art algorithms, followed by a discussion connecting MOSS to related works.

2 Problem Formulation

Wefi rst overview why stability is crucial in trustworthy and interpretable machine learning and introduce how we assess the stability of rule sets. Following this, we discuss stability selection, a popular statistical framework for improving model stability. We show how an optimization reinterpretation of this framework motivates our formulation of MOSS.

2.1 Model Stability

Stability has long been considered to be a prerequisite for trustworthiness when interpreting models [33, 35]. At the minimum, reliable conclusions drawn from models must be replicable across repeated analyses, a foundational idea in scientific discovery [27]. From a statistical perspective, we are interested in the stability of the results of repeated analyses conducted across datasets that differ by reasonable perturbations [33]. We formalize this in the context of constructing rule sets below.

We want to assess if an algorithm constructs the same set of rules when applied across different data perturbations. Say that we have datasets X_i and X_j that differ by some reasonable perturbation, for example, these two datasets may come from different training-test partitions of the original dataset X. Using the same algorithm, we construct rule sets R_i and R_j on each dataset. To measure the similarity between the two rule sets we use the Dice-Sorensens coefficient, defined by:

$$\text{DSC}(R_i, R_j) = \frac{2|R_i \cap R_j|}{|R_i| + |R_j|}. \tag{1}$$

This coefficient captures the proportion of decision rules shared between rule sets and is commonly used in assessing the stability of statistical models [26].

To assess the stability of our rule algorithm, we generate many perturbed datasets $X_1 \dots X_T$ and apply our algorithm to construct rule sets $R_1 \dots R_T$. We report the average pairwise Dice-Sorensens coefficient across all rule sets as the *empirical stability* of our algorithm, given by:

$$\text{Empirical Stability} = \frac{T(T-1)}{2} \sum_{i \neq j} \text{DSC}(R_i, R_j). \tag{2}$$

This metric captures the average proportion of rules shared between rule sets constructed across data perturbations. As an aside, in §H of the appendix, we discuss alternative metrics for rule stability and demonstrate that the results of our paper remain consistent across these metrics.

In the next section, we discuss how stability selection framework can be used to improve the stability of rule-based algorithms.

2.2 Stability Selection

The goal of stability selection is to remove the unstable components of a model. Given training dataset $X \in \mathbb{R}^{n \times p}$, the procedure starts by generating perturbed datasets $X_1 \dots X_B$ by sub-sampling rows or the bootstrap. On each dataset, we apply the sample algorithm to generate rule sets $R_1 \dots R_B$. Let m denote the total number of *unique* rules constructed across all sets. For each unique rule r_i for $i \in [m]$ we compute the proportion of rule sets in which r_i appears. We store these selection proportions in vector $\Pi \in R^m$. Stability selection extracts the rules with selection proportions above threshold τ, $\{r_i | \Pi_i > \tau\}$; these rules are considered the stable components of the model. By pruning the unstable components, stability selection improves the empirical stability of thefi nal model.

As an aside, the SIRUS algorithm mentioned in the introduction cleverly applies stability selection to the construction of rule sets by

leveraging random forests [3]. Random forests consist of decision treesfi t on bootstrapped datasets, and the leaves of each decision tree form a set of decision rules. SIRUS extracts the rules that appear consistently across trees in the random forest.

Threshold τ is a hyperparameter that controls the sparsity k of the selected model. For rule sets, k is often prespecified to be a managable number of rules (at most 15-20) to preserve interpretability [3, 21, 31]. Next, we show that we can reinterpret stability selection as an optimization problem for afi xed model size k.

2.3 Optimization Reinterpretation

Stability selection eliminates unstable rules by discarding those with selection proportions below threshold τ. For a predetermined model size of k rules, discarding unstable rules can be achieved by selecting rules with the top-k selection proportions Π. We formalize this task below.

Given a set of $r_1 \dots r_m$ unique decision rules with selection proportions $\Pi \in \mathbb{R}^m$, we introduce binary decision variables $z_i \in \{0, 1\}$, for $i \in [m]$, to indicate if rule r_i is selected. Selecting the top-k rules with the largest selection proportions can be expressed by this optimization problem:

$$\max_{z_1,\dots,z_m} H_1(z) = \sum_{i=1}^{m} \Pi_i z_i \quad \text{s.t.} \quad \sum_{i=1}^{m} z_i \le k, z \quad \in \{0,1\}^m. \tag{3}$$

We denote objective $H_1(z)$ as the *in-sample stability* of solution z, i.e., the sum of selection proportions of the selected rules. For a fixed model size k, stability selection maximizes in-sample stability as a proxy for the empirical stability of the model.

Problem 3 is a binary knapsack optimization problem and optimal solution z^* can be trivially computed by taking the k-largest elements of Π. Moreover, we can also compute a sequence of t solutions of progressive suboptimality by sorting Π in descending order sliding a window of length k along the sorted vector. This yields a sequence of solutions $z^*, z^1, z^2, \dots, z^t$ with decreasing in-sample stability objectives $H_1(z^*) > H_1(z^1) > H_1(z^2) \dots > H_1(z^t)$. Below, we show that exploring these sequences of suboptimal solutions yield important insights towards the trade-off between model accuracy and stability, insights that motivate the development of our MOSS framework.

2.4 Accuracy of Suboptimal Solutions

In this section, we examine a sequence of solutions $z^*, z^1, \dots, z^t$ to Problem 3 that are progressively suboptimal with respect to in-sample stability. Our goal is to explore how these solutions perform in terms of model accuracy, which we define for solution z below.

Model Accuracy: Given a set of decision rules $r_1 \dots r_m$ where $f_i(X) \in \mathbb{R}^n$ is the prediction of rule r_i, solution vector $z \in \{0, 1\}^m$ indicates which rules are selected. We want to assess how well the predictions of this selected rule setfi t the data. Predictions of a rule set are determined by a linear combination of the rules $\sum_{i=1}^{m} w_i f_i(X)$, where decision variable $w_i \neq 0$ if and only if $z_i \neq 0$, for all $i \in [m]$. Given response $y \in \mathbb{R}^m$ and solution vector z, we use regularized in-sample training loss to measure model accuracy,

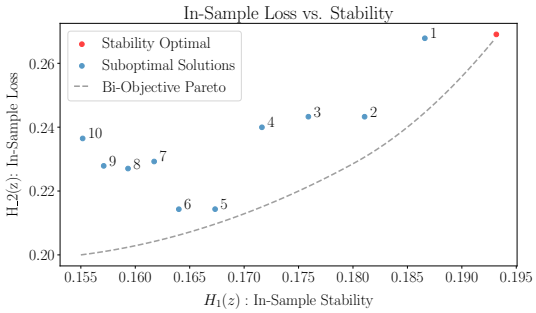

Figure 2: We apply stability selection to extract compact rule sets from 10^3 candidate rulesfi t on the ESL dataset [30]. Solutions suboptimal in terms of stability have much lower in-sample loss. The goal of MOSS is to compute the bi-objective Pareto front between accuracy and stability (gray line).

which is given by:

$$H_2(z) = \min_{w_1,\dots,w_n} \frac{1}{2}\|y - \sum_{i=1}^{m} f_i(X) w_i\|_2^2 + \frac{1}{2\gamma}\|w\|_2^2, \tag{4}$$
$$\text{s.t.} \quad w_i(1 - z_i) = 0 \quad \forall\, i \in [m], \quad z \in \{0,1\}^m.$$

We add the ridge penalty for computational reasons, however, we note that the regularizer may improve performance in noisy settings [23]. Hyperparameter $\gamma > 0$ controls the degree of shrinkage over w. Note that for any z, the optimal solution to the minimization problem in expression 4, w^*, can be obtained in closed-form through a back-solve. We show later in §3.1.2 that we can rewrite expression 4 in terms of just z.

Now that we have defined a way to assess model accuracy, through in-sample loss $H_2(z)$, we can begin our exploration. Using the procedure discussed in §2.3, we apply Problem 3 to construct rule sets on the ESL dataset [30]. We start with a collection of 10^3 decision rules generated from a random forest and, for afi xed set of $k = 20$ decision rules, we compute the stability selection optimal solution z^* and a sequence of 10 progressively suboptimal solutions $z^1, z^2, \dots, z^{10}$. For each solution, we compute in-sample stability $H_1(z)$ and in-sample loss $H_2(z)$.

In Figure 2, we plot in-sample loss against in-sample stability for all of the solutions. The red point in the top right of the plot shows the optimal stability selection solution z^* and the blue points show suboptimal solutions $z^1 \dots z^{10}$. We observe from this plot that solutions that are slightly suboptimal with respect to in-sample stability may have significantly lower in-sample loss.

This observation motivates MOSS. The goal of our framework is to efficiently compute the Pareto frontier between in-sample stability $H_1(z)$ and in-sample loss $H_2(z)$, as hypothesized by the gray dashed line in Figure 2. Importantly, we demonstrate later in §4.2 that choosing solutions along this Pareto that are suboptimal with respect to in-sample stability can significantly improve the out-of-sample predictive performance of the model without compromising out-of-sample empirical stability of the model. We present our MOSS framework below.

3 MOSS Framework

In this section we present the MOSS framework. Wefi rst discuss our multi-objective optimization formulation that accounts for both stability and accuracy. The optimization programs in this formulation are NP-hard, so we develop a specialized cutting plane-based optimization algorithm to solve problems in MOSS efficiently. Moreover, we develop a novel technique that exploits problem structure to efficiently compute the entire Pareto frontier between stability and accuracy.

3.1 Multi-Objective Optimization Formulation

Given data $X \in \mathbb{R}^{n\times p}$ and target $y \in \mathbb{R}^n$, the goal of MOSS is to construct rule sets that are both accurate and stable. Wefi rst follow the stability selection (SIRUS) framework and construct a large collection of unique candidate rules r_i for $i \in [m]$, with selection proportions Π. Our goal is to select which candidate rules to include into thefi nal rule set, so we let $z \in \{0,1\}^m$ represent a vector of binary decision variables that indicate which rules are selected.

Recall that in §2.3 and §2.4, we define in-sample stability $H_1(z)$ as a proxy for empirical stability and in-sample loss $H_2(z)$ as a proxy for out-of-sample accuracy. The goal of MOSS is tofit z to *maximize* in-sample stability and *minimize* in-sample loss. We express maximizing $H_1(z)$ as minimizing $-H_1(z)$ and formalize this in the problem below. MOSS uses the following bi-objective integer program to select a rule set of k decision rules:

$$\min_z \ -H_1(z), \quad H_2(z) \qquad \text{s.t.} \quad \sum_{i=1}^m z_i \le k, z \quad \in \{0,1\}^m. \tag{5}$$

The goal of our framework is to compute the Pareto frontier between objectives $H_1(z)$ and $H_2(z)$. We apply the ϵ-constraint method to accomplish this.

3.1.1 ϵ-Constraint Method: The ϵ-constraint method computes the Pareto frontier of a bi-objective optimization problem by solving a sequence of single-objective problems. The method involves moving one objective into the constraint set and constraining that objective to be more extreme than afi xed value ϵ. We sweep through values of ϵ while solving the corresponding single-objective problem to compute the Pareto frontier. This method is advantageous in that it can recover non-convex Pareto frontiers [11].

For MOSS, we move stability objective $-H_1(z)$ into the constraint set and constrain $-H_1(z)$ to be less than or equal to $-\epsilon$, where ϵ is non-negative. We sweep through values of ϵ while solving this single-objective integer program:

$$\min_z \ H_2(z) \quad \text{s.t.} \quad H_1(z) \ge \epsilon, \quad \sum_{i=1}^m z_i \le k, z \quad \in \{0,1\}^m. \tag{6}$$

to compute our Pareto frontier. By plugging in our definition for model accuracy objective $H_2(z)$, expression 4, we can express this problem as the following mixed integer program:

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{2}\|y - \sum_{i=1}^m f_i(X) w_i\|_2^2 + \frac{1}{2\gamma}\|w\|_2^2, \\ \text{s.t.} \quad & \sum_{i=1}^m \Pi_i z_i \ge \epsilon, \quad \sum_{i=1}^m z_i \le k, \\ & w_i(1-z_i) = 0 \quad \forall\, i \in [m], \quad z \in \{0,1\}^m. \end{aligned} \tag{7}$$

We show below that we can reformulate this optimization problem entirely in terms of z.

3.1.2 Binary Integer Reformulation: In Proposition 1, we show that we can reformulate Problem 7 into a binary integer program by re-expressing the objective in terms of z.

Proposition1. *Let prediction matrix $M \in \mathbb{R}^{n\times m}$ contain predictions $f_i(X) \in \mathbb{R}^n$ in column i, for $i \in [n]$. We can re-write our expression for regularized in-sample loss, $H_2(z)$, given solution vector z as:*

$$H_2(z) = \frac{1}{2} y^\top \left(\mathbb{I}_n + \gamma \sum_{i=1}^m z_i M_i M_i^\top \right)^{-1} y, \tag{8}$$

where M_i is the i-th column of M.

We show this derivation in the appendix (A.1). As such, we can rewrite Problem 7 as:

$$\begin{aligned} \min_z \quad & \frac{1}{2} y^\top \left(\mathbb{I}_n + \gamma \sum_{i=1}^m z_i M_i M_i^\top \right)^{-1} y, \\ \text{s.t.} \quad & \sum_{i=1}^m \Pi_i z_i \ge \epsilon, \quad \sum_{i=1}^m z_i \le k, z \quad \in \{0,1\}^m. \end{aligned} \tag{9}$$

This binary integer program is the main optimization problem that we use to construct rule sets in MOSS, where ϵ and k are non-negative hyperparameters that control the stability and sparsity of the model respectively.

Problem 9 has m decision variables and $m + 2$ constraints; m represents the size of the candidate rule set, which can be very large. For example, the SIRUS approach uses random forests to generate thousands of candidate rules. As such, instances of Problem 9 are often intractable and exceed the capabilities of off-the-shelf optimization software, as we show in §4.1. We can, however, exploit problem structure to develop a tailored optimization algorithm to solve this problem efficiently. Our specialized algorithm hinges on the fact that we move stability objective $H_1(z)$ into the constraint set, and not accuracy objective $H_2(z)$. By moving $H_1(z)$, we obtain constraints that are linear with respect to z. Also, the combinatorial space of feasible solutions for z shrink as ϵ increases. We exploit these properties in our algorithm below.

3.2 Cutting Plane Algorithm

Here, we develop a specialized cutting plane algorithm to efficiently solve Problem 9 to optimality. Our algorithm leverages the fact that while integer programs with nonlinear objectives like Problem 9 are often intractable, integer linear programs (ILPs) with linear objectives can be efficiently solved using off-the-shelf methods [4]. Therefore, we reformulate the task of solving Problem 9 into solving a sequence of ILPs. Problem 9 is a binary integer program with linear constraints and we show in the proposition below that objective is convex.

Proposition2. *Function $H_2(z)$ is convex on domain $z \in \mathbb{R}^m$ such that $z_i \in [0,1]$, for all $i \in [m]$.*

We show the full proof of this proposition in the appendix (A.2). Since $H_2(z)$ is convex, we can apply an cutting plane algorithm in order to solve Problem 9 efficiently [4, 10]. Let v represent the lower bound on objective value $H_2(z)$. At each iteration t of our cutting plane algorithm, wefi rst add cutting plane $v \geq H_2(z^t) + \nabla H_2(z^t)(z - z^t)$. The approximation function at iteration t is given by $\max_{j \in [t]} H_2(z^j) + \nabla H_2(z^j)(z - z^j)$. We then minimize this function, with respect to the linear constraints in Problem 9 to update v and z for the subsequent iteration. This minimization involves solving an integer linear program, which can be done efficiently using off-the-shelf methods. We terminate our algorithm when $H_2(z^t) \leq v$ and our algorithm terminates after afi nite number of iterations and returns the optimal solution to Problem 9. We discuss convergence further in §A.3 of the appendix. Our full cutting plane algorithm is presented below.

Algorithm 1: Cutting Plane

```
z^0 ← warm start
v^0 ← H_2(z^0), t ← 0
while H_2(z^t) > v do
    add constraint
        v ≥ H_2(z^t) +
        ∇H_2(z^t)(z − z^t) to
        ILP (Problem 10)
    solve ILP for v^{t+1}, z^{t+1}
    t = t+1
end
return z*
```

Problem 10: Integer Linear Program (ILP)

$$\min_{v,z} \quad v \tag{10a}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} \Pi_i z_i \geq \epsilon, \tag{10b}$$

$$\sum_{i=1}^{m} z_i \leq k, z \quad \in \{0,1\}^m, \tag{10c}$$

$$v \geq H_2(z^j) + \nabla H_2(z^i)(z - z^j), \quad \forall\, j \in [t]. \tag{10d}$$

In each iteration of our cutting plane algorithm (Algorithm 1), we must solve integer linear program Problem 10. Off-the-shelf solvers can typically solve this problem in seconds, for problem sizes on the order of $m \sim 10^5$ decision variables. Below, we discuss two attributes of Algorithm 1 that further improve computational efficiency.

3.2.1 Sandwiching Constraints: Problem 10 is especially easy to solve, for afi xed k, when ϵ is large, since constraints 10b and 10c work together to sandwich down the combinatorial space of feasible solutions for z. We show this effect in Figure 3, as ϵ increases the time it takes for an off-the-shelf optimization solver (Gurobi) to solve ILP Problem 10 drastically decreases. As such, Algorithm 1 is especially efficient when ϵ is large.

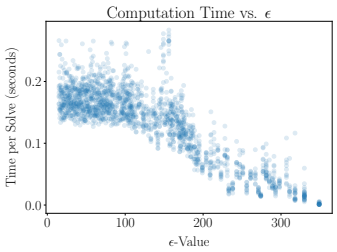

Figure 3: Computation time per solve for ILP plotted against ϵ, for fixed sparsity $k = 20$.

3.2.2 Efficient Gradient and Objective Evaluation: In each iteration of Algorithm 1, we must evaluate objective $H_2(z)$ and compute gradient $\nabla H_2(z)$ for solution z. We show here that these computations can be conducted efficiently. Given solution vector z, let κ represent the set of nonzero indices in z. The cardinality of set κ is restricted to be at most k, the number of decision rules constructed in the rule set. Furthermore, let M_κ represent the sub-matrix of M with the κ-indexed columns selected. Computing gradient $\nabla H_2(z)$ and evaluating objective $H_2(z)$ is bottle-necked by the matrix inversion $(\frac{\mathbb{I}}{\gamma} + M_\kappa^\top M_\kappa)^{-1}$, i.e., the cost of inverting a square matrix of at most size $k \times k$ [4]. Since we are interested in using MOSS to construct a sparse interpretable rule set, of no more than 10-20 decision rules, k is typically small and this computation is cheap.

Algorithm 1, with the attributes discussed above, allows us to solve Problem 9 efficiently for afi xed value of ϵ; we show timing results in §4.1. The algorithm is especially efficient when ϵ is large and when k is small; k is typically kept small in MOSS for interpretability reasons. However, we must solve Problem 9 across a range of ϵ values to compute the Pareto frontier between accuracy and stability. We discuss how to do so efficiently below.

3.3 Computing the Pareto Frontier

In this section, we introduce our method to efficiently compute the Pareto frontier between stability objective $H_1(z)$ and accuracy objective $H_2(z)$. To compute this Pareto frontier, we must repeatedly solve Problem 9 across a range of ϵ values, and wefi rst discuss what values of ϵ to solve for below.

3.3.1 ϵ-Range: We demonstrate here that it is sufficient to solve Problem 9 for a discrete sequence of ϵ values to fully compute the Pareto frontier between $H_1(z)$ and $H_2(z)$ with complete granularity. First, we note that forfi xed sparsity k, the largest value of ϵ such that Problem 9 is feasible is equivalent to the k largest elements of Π. We denote this value as $\epsilon_{\max}$ and recall that from §2.3 that $\epsilon_{\max}$ is equivalent to the optimal objective value of our stability selection optimization formulation (Problem 3).

To compute the Pareto frontier, we are interested infi nding a sequence of ϵ-values that correspond to a sequence of increasingly suboptimal objectives for $H_1(z)$. As discussed in §2.3, this can be easily obtained by sorting Π in descending order and taking the sum of a rolling window of k elements down the vector to obtain objective values $H_1(z^*) > H_1(z^1) > \ldots > H_1(z^t)$, where $t = m - k + 1$. We set these objective values as the sequence $\mathbb{E} = \epsilon_{max} > \epsilon^1 > \ldots > \epsilon^t$. To compute the entire Pareto frontier between $H_1(z)$ and $H_2(z)$, in full granularity, we solve Problem 9 for every value of $\epsilon \in \mathbb{E}$. It is important to note that we do not necessarily need to compute the Pareto frontier between $H_1(z)$ and $H_2(z)$ in full granularity. Rather, we can often compute subsequence or interesting segments of $\mathbb{E}$ tofi nd an appropriate model.

3.3.2 Efficient Pareto Method: Given a descending sequence of ϵ-values $\mathbb{E}$, we develop the following algorithm to efficiently solve Problem 9 for each value of $\epsilon \in \mathbb{E}$. Our algorithm relies on the fact that we can exploit nested problem structure to reuse cutting planes, which greatly reduces the number of iterations needed for Algorithm 1 to converge.

Say that we solve Problem 9 for two values of $\epsilon \in \mathbb{E}$, where $\epsilon_1 > \epsilon_2$. Any solution to Problem 9 with $\epsilon = \epsilon_1$ is also a feasible solution to Problem 9 with $\epsilon = \epsilon_2$. As such, the cutting planes and approximation function generated when solving the ϵ_1 problem are also valid for the ϵ_2 problem. If we start with $\epsilon_{\max}$ and solve down the sequence $\mathbb{E}$, we are solving a sequence of nested optimization problems, where every feasible solution found is also feasible for the subsequent problem. Consequently, each time we solve Problem 9 using Algorithm 1, we can re-use all of the cutting planes from earlier problems. This greatly reduces the number of iterations of Algorithm 1 for each solve in the sequence. We present this efficient Pareto method (EPM) in the algorithm below.

Algorithm 2: Efficient Pareto Method (EPM)

Given: $\mathbb{E}$, k
$Z \leftarrow \emptyset$ // Set of solutions.
$C \leftarrow \emptyset$ // Set of cutting planes.
$t = 0$, $z_t = \emptyset, c_t = \emptyset$ // Solution and cutting planes at iteration t.
while $\epsilon \in \mathbb{E}$ **do**
  Warm start Algorithm 1 with constraint set C and current solution z_t.
  Apply Algorithm 1 for ϵ and k to obtain solution z_{t+1} and cutting planes c_{t+1}.
  $Z \leftarrow Z \bigcup z_t$
  $C \leftarrow C \bigcup c_t$
  $t = t + 1$
end
return Z

In practice, EPM greatly reduces the number of iterations required each time we apply Algorithm 1 when we compute the Pareto frontier. We show this though an example below.

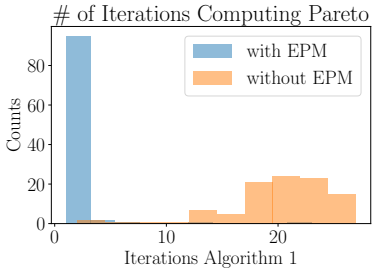

Figure 4: Our efficient EPM procedure reduces the number of OA iterations needed to compute the Pareto frontier.

We use MOSS with and without EPM to compute thefi rst 100 values of $\mathbb{E}$. In Figure 4, we show the number of iterations required each time we apply our cutting plane Algorithm 1. This corresponds to the number of cutting planes added, and the number of times we need to solve ILP Problem 10. We see that EPM greatly reduces the number of iterations required, fact, often times only a few additional cutting planes are needed to solve Problem 9 for the next value of ϵ. In §4.1, we show that this translates to large computational savings.

3.4 Putting Together the Pieces

By combining our new multi-objective optimization formulation in MOSS with our tailored cutting plane algorithm, we can efficiently explore the Pareto frontier between in-sample stability $H_1(z)$ and in-sample loss $H_2(z)$. Recall that we maximize in-sample stability as a proxy for empirical out-of-sample stability and minimize in-sample loss as a proxy for out-of-sample error. In this section, we apply MOSS to an example to explore how solutions along the Pareto frontier of $H_1(z)$ and $H_2(z)$ perform in terms empirical stability and out-of-sample $1 - R^2$.

On the Galaxy dataset from OpenML [30], wefi rst apply the SIRUS framework to generate a set of $m = 1000$ unique decision rules with selection proportions Π. Wefi x the sparsity of our desired rule set at $k = 15$ and apply MOSS to compute the entire Pareto frontier between in-sample stability and in-sample accuracy.

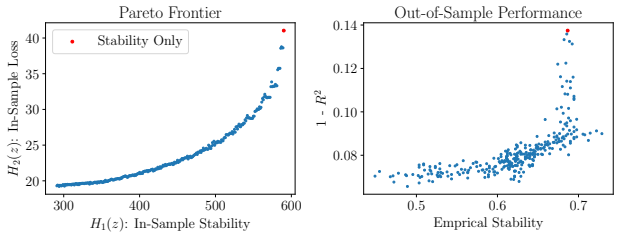

Figure 5: Left Panel: Pareto frontier between accuracy and stability recovered by MOSS. Note the steepness of the Pareto front near the stability-only solution. Right Panel: Out-of-sample accuracy and stability for MOSS solutions. MOSS can find stable solutions with much improved test accuracy.

The left panel in Figure 5 displays the Pareto frontier. The horizontal axis shows in-sample stability and the vertical axis shows in-sample loss, and the points show solutions of MOSS for different values of ϵ. The red point at the top right of the Pareto indicates the solution when ϵ is set to its maximum value, meaning MOSS optimizes solely for in-sample stability. Note that the Pareto frontier drops sharply directly to the left of the red point. This is important; many solutions on the Pareto frontier slightly suboptimal in terms of in-sample stability have much lower in-sample loss.

Importantly, these solutions demonstrate better out-of-sample predictive performance while preserving good out-of-sample empirical stability. In the right panel in Figure 5, we repeat the procedure detailed above across a 10-fold CV of the Galaxy dataset. On the horizontal axis, we report out-of-sample empirical stability, defined via the average DSC metric presented in §2.1, and on the vertical axis we report out-of-sample performance, via 1-R^2, averaged across all folds. Again, each point shows MOSS solutions for different values of ϵ. We observe that along the right hand side of the plot, many solutions have high empirical stability but some have much better out-of-sample performance; these correspond to the solutions that are slightly suboptimal in terms of in-sample stability discussed in the paragraph above.

We hypothesize that solutions in MOSS that are slightly suboptimal in terms of in-sample stability mayfi t the data much better, and produce models with much better out-of-sample performance. Moreover, these solutions may not necessarily exhibit worse out-of-sample stability, as we see above. Finally, as discussed in §3.2, the cutting plane algorithm in MOSS is extremely efficient at exploring these solutions. We test this hypothesis in §4.2 by evaluating how well MOSS performs compared to several state-of-the-art rule-based algorithms in terms of both accuracy and stability.

3.5 Approximate Algorithm

As an aside, we also present an approximate algorithm tofi nd high-quality solutions to Problem 9. Our algorithm is self-contained and is much more computationally efficient than our outer-approximation approach when ϵ is small.

Wefi rst re-express Problem 9 in terms of just w, where $z_i = \mathbb{I}(w_i \neq 0)$, and work with the unconstrained Lagrangian of this problem:

$$\min_w \frac{1}{2}||y-Mw||_2^2+\frac{1}{2\gamma}||w||_2^2+\lambda_1 \sum_{i=1}^{m} \mathbb{1}(w_i \neq 0)-\lambda_2 \sum_{i=1}^{m} \Pi_i \mathbb{1}(w_i \neq 0).$$

We drop the constraints that involve k and ϵ and use non-negative hyperparameters λ_1 and λ_2 to control the sparsity and in-sample stability of the extracted rule sets. This expression simplifies to:

$$\min_w \frac{1}{2}\left(||y-Mw||_2^2+\frac{1}{\gamma}||w||_2^2\right)+\left(\sum_{i=1}^{m} \mathbb{1}(w_i \neq 0)(\lambda_1-\Pi_i\lambda_2)\right), \quad (11)$$

where thefi rst term is smooth and the second term is separable over w. As such, we apply coordinate descent to Problem 11 tofind good solutions, since the algorithm will converge to a local minima. Each coordinate update has a closed-form solution, and we present our full coordinate descent heuristic in the appendix (B).

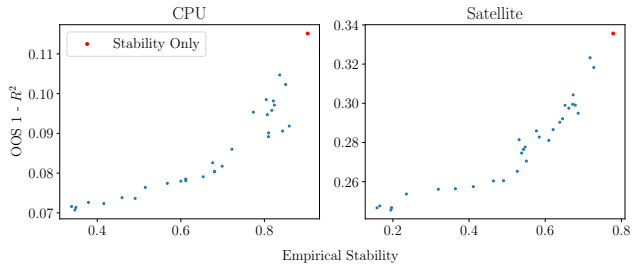

Figure 6: Our approximate algorithm recovers good approximations of the Pareto frontier between accuracy and stability.

In practice, we observe that our approximate algorithm can efficiently compute high-quality approximations of the Pareto frontier between accuracy and stability. In Figure 6, we use our approximate algorithm to construct rule sets of sparsity $k = 15$ on the CPU and Satellite datasets from OpenML [30]. We sweep though parameter λ_2 to explore the trade-off between accuracy and stability, and we plot out-of-sample $1 - R^2$ against empirical stability. We see from thisfi gure that our approximate algorithm canfi nd good solutions that balance out-of-sample accuracy with empirical stability.

4 Experiments

We evaluate the computation time and performance of MOSS.

4.1 Computation Time Experiment

Wefi rst evaluate our cutting plane algorithm (Algorithm 1) against two state-of-the-art commercial solvers, Gurobi and MOSEK [1, 16]. Using each method we solve various instances of Problem 9 across 10fi xed values of ϵ, for rule sparsity $k = 15$. We conduct this timing experiment on a 2022 M2 Macbook Pro.

We show these computation time results in Table 1. The leftmost column shows the problem size in terms of the number of data points n and the number of decision variables m. We observe that on the smallest problems our cutting plane algorithm achieves orders of magnitude speedups compared to commercial solvers. For larger problems, we observe that our cutting plane algorithm scales well, and can handle problem sizes beyond the capabilities of off-the-shelf optimization solvers.

Data Points / Variables	Cutting Plane	Gurobi	MOSEK
150, 250	0.044s (0.005)	28m 15s	40m 12s
150, 500	0.0965s (0.003)	31m 10s	45m 6s
150, 1000	0.464s (0.68)	1h 30m	1h 42m
1500, 1200	0.66s (0.09)		
5000, 500	3.07s (0.1)		
7500, 1500	2.27s (0.9)		
7500, 2500	2.3s (0.5)		
15000, 700	14.1s (0.3)		

Table 1: Computation time results, the red cells indicate that the method fails to return the optimal solution after 4 hours.

4.1.1 Ablation Study. We also investigate the impact of our efficient Pareto method on computation time. Our efficient Pareto method exploits nested problem structure to reuse cutting planes are warm-starts when computing the Pareto frontier, so we compare this method against naively sweeping through values of ϵ.

Data Points / Variables	w/ EPM	w/o EPM
1000, 500	13s (2.1)	26s (1.4)
3000, 600	65s (5.2)	5m 10s (24.1)
5000, 500	58s (4.3)	9m 12s (32.1)
8000, 400	9m 5s (10.2)	55m 12s (42.3)

Table 2: Timing results for ablation study.

On several instance of Problem 9 of varying sizes, we use MOSS to compute the Pareto frontier across 100 values of ϵ, with and without our efficient Pareto method (EPM). We show the results in Table 2. We see from this table that EPM drastically reduces the computation time required for MOSS to compute Pareto frontiers and can yield up to an order of magnitude speedup for larger problem sizes.

4.2 Performance Experiment

We evaluate MOSS against competing rule-based algorithms in terms of stability and predictive performance.

Experiment Procedure. We repeat this procedure on 30 regression datasets of various sizes sourced from OpenML [30]; the full lists of datasets with metada can be found in the appendix (D.1). On each dataset, we conduct a 10-fold cross validation and on each fold wefi t a random forest to generate $m \sim 10^3$ unique candidate rules with selection proportions Π. We apply MOSS to construct sets of $k = 15$ decision rules under three settings. For the high ϵ setting (**MOSS-ϵ-H**) we set ϵ to be the 3rd element in sequence $\mathbb{E}$, close to ϵ_{max}, and for the medium setting (**MOSS-ϵ-M**) we set ϵ to be the 40th element of $\mathbb{E}$, close to the midpoint of the sequence. For these two settings, we compute solutions using our OA algorithm with EPM. For the low setting (**MOSS-ϵ-L**) we compute solutions

using our approximate algorithm. Across all folds, we evaluate the out-of-sample empirical stability, defined via average pairwise DSC, and predictive performance, defined via average out-of-sample R^2, of the constructed rule sets.

We compare MOSS against the following state-of-the-art competing algorithms: **SIRUS** (2021) [3] which constructs decision rules with respect to stability only, **FIRE** (2023) [21] which selects rule sets using the non-convex MCP penalty, **GLRM** (2019)[31] which constructs compact rule sets from scratch using column generation, and **RuleFit** (2008) [14] which selects rules using the LASSO. For a fair comparison, we use each algorithm to construct rule sets of $k = 15$ decision rules. Again, we assess the out-of-sample empirical stability and predictive performance of all algorithms across all folds. Further details about our experiment can be found in the appendix (D.2).

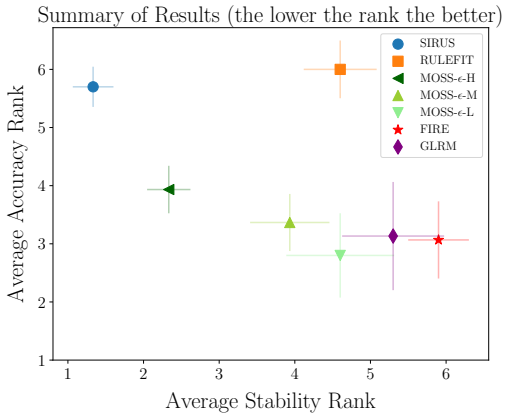

Figure 7: Method rank in terms of accuracy and stability.

Results. We discuss our experimental results; tables of the detailed results for each dataset can be found in the appendix (E). Wefirst consider the ranking of each method, in terms of accuracy and predictive performance, on each dataset in the experiment (with 1 being the best and 7 being the worst). In Figure 7, we plot the average rank of each method where the horizontal axis shows accuracy and the vertical axis shows stability. The points show which method is used and the lines show standard error. We observe from thisfi gure that SIRUS performs the best in terms of model stability but nearly the worst in terms of model accuracy. Among the competing methods, FIRE and GLRM perform the best in terms of accuracy but the worst in terms of stability. RuleFit performs the worst in terms of accuracy but is more stable than FIRE, which may be due to the added shrinkage of the LASSO [21].

Our MOSS methods are unique in that they perform well with respect to both objectives. MOSS-ϵ-H is the second most stable method, and is much more accurate compared to SIRUS and RuleFit. Interestingly, MOSS-ϵ-L performs the best in terms of model accuracy and is still more stable than FIRE and GLRM. This may be due to the fact that the combined sparsity-stability penalty in our approximate algorithm leads to better out-of-sample performance compared to penalizing sparsity only. As expected, MOSS-ϵ-M balances stability and accuracy compared to the competing methods.

To condense accuracy and stability into a single metric we take the average combined rank of each method across all dataset. We show these results in the table below and it is apparent that MOSS-ϵ-H performs the best in terms of this metric. As such, we recommend setting MOSS-ϵ-H as the default when using our framework.

MOSS-ϵ-H	MOSS-ϵ-M	MOSS-ϵ-L	SIRUS	FIRE	GLRM	RULEFIT
3.13	3.65	3.74	3.51	4.18	4.51	5.3

With this in mind, we compare MOSS-ϵ-H against SIRUS and FIRE, the competing algorithms that perform the best in terms of stability and accuracy individually. On 24 out of the 31 datasets in our experiment, the empirical stability of the SIRUS rule set is within one standard error of MOSS-ϵ-H, making rule sets functionally identical in terms of stability. On these datasets, MOSS-ϵ-H is much more accurate and exhibits an average **10%** increase in out-of-sample R^2. Compared to FIRE, MOSS-ϵ-H exchanges a **2%** decrease in out-of-sample R^2 for a **190%** increase in empirical stability. We show the distribution of these percent differences in the appendix (F). As our experiments show, MOSS can construct rule sets that jointly out-perform our competing methods in terms of both accuracy and stability.

5 Conclusion

We conclude by showcasing the utility of MOSS on two real-world case studies. We also discuss connections between MOSS and related works and analyze the sensitivity of our framework to parameters γ and k.

Case Studies. In Section C of the appendix, we present two case studies where we apply MOSS to real-world problems: scientific discovery in marine biology and census planning. In both of these examples, we show that MOSS can construct stable rule sets that perform well and reveal uncover insights about the data.

Related Work. We explore connections between MOSS and Rashomon set algorithms in interpretable machine learning [9, 22, 32]. Rashomon set algorithms identify a collection of models—such as decision rules—that achieve near optimal predictive performance in terms of training error. Despite their similar accuracy, these models can vary significantly in other aspects, such as fairness or the features they use. By analyzing this so-called Rashomon set of near optimal models, practitioners can account for these additional considerations when selecting an appropriate.

Consider the task of extracting rule sets from tree ensembles. Under this setting, we see an example of the Rashomon effect in Figure 2. In this plot, the extracted rule sets indicated by the blue points labeled 2 through 10 have similar in-sample training errors, however, the in-sample stability of these models vary significantly.

With this in mind, we can consider an alternative formulation of MOSS that explores the Rashomon set of size-k rule sets that can be extracted from a tree ensemble, and selects the rule set with the highest in-sample stability. This formulation is given by:

$$\begin{aligned} \min_{w,z} \quad & -H_1(z), \\ \text{s.t.} \quad & \frac{1}{2}\|y - \sum_{i=1}^{m} f_i(X) w_i\|_2^2 \le L^* + \psi, \quad \sum_{i=1}^{m} z_i \le k, \\ & w_i(1 - z_i) = 0 \quad \forall\, i \in [m], \quad z \in \{0,1\}^m, \end{aligned} \tag{12}$$

where L^* is the lowest possible training error achieved by extracting a size-k rule set from the tree ensemble and ψ is the error tolerance [32]. Problem (12) is similar to MOSS if we were to move objective $H_2(z)$ into the constraint set for our ϵ-constraint method. However, as we discuss in §3.1.2, moving the in-sample loss objective into the constraint set results in a much more difficult problem to solve due to its non-linearity. As such, MOSS explores the collection of size-k rule sets with near optimal in-sample stabilities and selects the model with the lowest in-sample loss. This is similar to a Rashomon set method, but it leads to a form more amenable to optimization.

It is important to note that while Rashomon set methods are interesting, enumerating or exploring the Rashomon set is extremely computationally challenging. As such, current Rashomon set methods for decision rules are restricted to binary classification tasks [9, 32] which prevents direct comparisons with our method. MOSS explores the Pareto frontier of just two objectives, accuracy and stability, extremely efficiently so that practitioners can rapidly select an appropriate model.

Sensitivity Analyses. In Section G of the appendix, we evaluate the sensitivity of MOSS to two parameters: γ, which controls the regularization penalty in the accuracy objective, and k, which controls the size of the constructed rule sets. In G.1, we demonstrate that MOSS is relatively *insensitive* to γ in terms of both accuracy and stability. In G.2, we show that increasing k improves the accuracy and stability of rule sets, but reduces interpretability. Additionally, we show results for our experiment in §4.2 for different values of k. Based on thesefi ndings, we recommend selecting $\gamma \in [10^{-3}, 10^{-2}]$ and $k \in \{10, 15, 20\}$ as good starting points for MOSS.

Final Remarks. MOSS is a useful framework that quickly constructs rule sets to explore the Pareto frontier between stability and accuracy. By examining this trade-off, practitioners can select models suitable for drawing trustworthy insights from the data.

Acknowledgments. We gratefully acknowledge support from the Office of Naval Research and the MIT Sloan Health Systems Initiative.

References

[1] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019. URL http://docs.mosek.com/9.0/toolbox/index.html.

[2] Vijay Arya, Rachel K. E. Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Q. Vera Liao, Ronny Luss, Aleksandra Mojsilović, Sami Mourad, Pablo Pedemonte, Ramya Raghavendra, John Richards, Prasanna Sattigeri, Karthikeyan Shanmugam, Moninder Singh, Kush R. Varshney, Dennis Wei, and Yunfeng Zhang. One explanation does notfi t all: A toolkit and taxonomy of ai explainability techniques, 2019. URL https://arxiv.org/abs/1909.03012.

[3] Clément Bénard, Gérard Biau, Sébastien Da Veiga, and Erwan Scornet. Sirus: Stable and interpretable rule set for classification. *Electronic Journal of Statistics*, 15:427–505, 2021.

[4] Dimitris Bertsimas and Bart Van Parys. Sparse high-dimensional regression. *The Annals of Statistics*, 48(1):300–323, 2020.

[5] Dimitris Bertsimas, Jean Pauphilet, and Bart Van Parys. Sparse classification: a scalable discrete optimization perspective. *Machine Learning*, 110:3177–3209, 2021.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[7] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[8] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[9] Martino Ciaperoni, Han Xiao, and Aristides Gionis. Efficient exploration of the rashomon set of rule-set models. In *Proceedings Of The 30th ACM SIGKDD Conference On Knowledge Discovery And Data Mining*, pages 478–489, 2024.

[10] Marco A Duran and Ignacio E Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical programming*, 36: 307–339, 1986.

[11] Michael TM Emmerich and André H Deutz. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing*, 17:585–609, 2018.

[12] Chandra Erdman and Nancy Bates. The low response score (lrs) a metric to locate, predict, and manage hard-to-survey populations. *Public Opinion Quarterly*, 81(1): 144–156, 2017.

[13] Roger Fletcher and Sven Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical programming*, 66:327–349, 1994.

[14] Jerome H Friedman and Bogdan E Popescu. Predictive learning via rule ensembles. 2008.

[15] Tyler O Gagné, K David Hyrenbach, Molly E Hagemann, Oron L Bass, Stuart L Pimm, Mark MacDonald, Brian Peck, and Kyle S Van Houtan. Seabird trophic position across three ocean regions tracks ecosystem differences. *Frontiers in Marine Science*, 5:317, 2018.

[16] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

[17] Longfei Han, Senlin Luo, Jianmin Yu, Limin Pan, and Songjing Chen. Rule extraction from support vector machines using ensemble learning approach: an application for diagnosis of diabetes. *IEEE journal of biomedical and health informatics*, 19(2):728–734, 2014.

[18] Trevor Hastie, Junyang Qian, and Kenneth Tay. An introduction to glmnet. *CRAN R Repositary*, 5:1–35, 2021.

[19] Hussein Hazimeh, Rahul Mazumder, and Tim Nonet. L0learn: A scalable package for sparse learning using l0 regularization. *Journal of Machine Learning Research*, 24(205):1–8, 2023.

[20] Jae June Lee and Cara Brumfield. The 2020 census & the environment: How census data are used for environmental justice & climate action. November 2019. URL https://www.georgetownpoverty.org/issues/the-2020-census-and-the-environment/.

[21] Brian Liu and Rahul Mazumder. Fire: An optimization approach for fast interpretable rule extraction. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1396–1405, 2023.

[22] Kota Mata, Kentaro Kanamori, and Hiroki Arimura. Computing the collection of good models for rule lists. *arXiv preprint arXiv:2204.11285*, 2022.

[23] Rahul Mazumder, Peter Radchenko, and Antoine Dedieu. Subset selection with shrinkage: Sparse linear modeling when the snr is low. *Operations Research*, 71 (1):129–147, 2023.

[24] Nicolai Meinshausen and Peter Bühlmann. Stability selection. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 72(4):417–473, 2010.

[25] W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019.

[26] Sarah Nogueira, Konstantinos Sechidis, and Gavin Brown. On the stability of feature selection algorithms. *Journal of Machine Learning Research*, 18(174):1–54, 2018.

[27] Karl Popper. *The logic of scientific discovery*. Routledge, 2005.

[28] Cynthia Rudin. Algorithms for interpretable machine learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1519–1519, 2014.

[29] U.S. Census Bureau. Planning database, 2023. URL https://www.census.gov/data/developers/data-sets/planning-database.html. Accessed: 2025-02-09.

[30] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

[31] Dennis Wei, Sanjeeb Dash, Tian Gao, and Oktay Gunluk. Generalized linear rule models. In *International conference on machine learning*, pages 6687–6696. PMLR, 2019.

[32] Rui Xin, Chudi Zhong, Zhi Chen, Takuya Takagi, Margo Seltzer, and Cynthia Rudin. Exploring the whole rashomon set of sparse decision trees. *Advances in neural information processing systems*, 35:14071–14084, 2022.

[33] Bin Yu. Stability. 2013.

[34] Bin Yu. Three principles of data science: predictability, computability, and stability (pcs). 2018.

[35] Bin Yu. Veridical data science. In *Proceedings of the 13th international conference on web search and data mining*, pages 4–5, 2020.

Appendix

Sections A, B, C, D of the appendix can be found below. Sections E, F, G, H, and I of our appendix can be found in this online supplement.

Code to reproduce our experiments can be found in this repository: github.com/brianliu12437/MOSS

A Proofs

We show the proofs for our propositions below.

A.1 Proof of Proposition 1. First, given decision vector z, let s be the set of indices such that $z \neq 0$. Let matrix M_s correspond to the sub-matrix of M with the s indexed rows. We have that:

$$\sum_{i=2}^{m} z_i M_i M_i^\top = M_s (M_s)^\top. \tag{1}$$

We start with function:

$$H_2(z) = \min_{w_1,\ldots,w_n} \frac{1}{2}\|y - Mw\|_2^2 + \frac{1}{2\gamma}\|w\|_2^2 \tag{2}$$

We have that minimizer $w^* = \left(\frac{\mathbb{I}}{\gamma} - M^\top M\right)^{-1} M^\top y$. Plugging in w^* into $\frac{1}{2}\|y - Mw\|_2^2 + \frac{1}{2\gamma}\|w\|_2^2$ evaluating the objective, and combining with the expression above, yields the desired expression for $H_2(z)$.

A.2 Proof of Proposition 2. We start with function:

$$H_2(z) = \frac{1}{2} y^\top \left(\mathbb{I}_n + \gamma \sum_{i=1}^{m} z_i M_i M_i^\top\right)^{-1} y.$$

Let matrix $A = \left(\mathbb{I}_n + \gamma \sum_{i=1}^{m} z_i M_i M_i^\top\right)$. We can re-express $H_2(z) = f(A) = y^\top A^{-1} y$. Matrix A is symmetric positive definite so function f is convex over A (follows from example 3.4 [6]).

Define function $g(z) = \left(\mathbb{I}_n + \gamma \sum_{i=1}^{m} z_i M_i M_i^\top\right)$; function g is affine over z. Convex combinations of affine functions are convex so $H_2(z) = f(g(z))$ is convex completing the proof.

A.3 Convergence of Cutting Plane Algorithm. The convergence of our cutting plane algorithm (Algorithm 1) is given by Corollary 1.

Corollary1. *Algorithm 1 terminates after afi nite number of cutting planes and returns z^*, the optimal solution to Problem 9.*

This follows directly from Fletcher and Leyffer (1994) [13].

B Coordinate Descent Heuristic

In this section, we present our coordinate descent-based heuristic tofi nd good solutions to optimization problems in MOSS.

Our CD heuristicfi nds good solutions to the problem:

$$\min_{w} \frac{1}{2}\left(||y - Mw||_2^2 + \frac{1}{\gamma}||w||_2^2\right) + \left(\sum_{i=1}^{m} \mathbb{1}(w_i \neq 0)(\lambda_1 - \Pi_i \lambda_2)\right).$$

We cycle through indices $k \in 1 \ldots m$ and update each decision variable w_k one-at-a-time.

Cyclic Updates: Given afi xed index k, let δ represent the set of remaining indices $\{1 \ldots m\} \setminus k$. We define the residual vector as $r_k = y - \sum_{i \in \delta} M_i w_i$. Each update in our coordinate descent algorithm aims to solve the problem:

$$\min_{w_k} \frac{1}{2}||r - M_k w_k||_2^2 + \frac{1}{2\gamma}(w_k)^2 + \mathbb{1}(w_k \neq 0)(\lambda_1 - \Pi_k \lambda_2). \tag{3}$$

We can solve this problem efficiently by considering two scenarios. If $w_k = 0$, we have that the objective value of Problem 3 is equal to $\frac{1}{2}||r||_2^2$. If $w_k \neq 0$, we have that:

$$w_k^* = \frac{M_k^\top r}{M_k^\top M_k + \frac{1}{\gamma}},$$

and that the objective is equal to

$$\frac{1}{2}||r - M_k w_k^*||_2^2 + \frac{1}{2\gamma}(w_k^*)^2 + \lambda_1 - \Pi_k \lambda_2.$$

We take the scenario with the lower objective value as the solution for Problem 3.

For our CD heuristic, we sweep through coordinates $k \in \{1 \ldots m\}$ while solving Problem 3, and repeat sweeps until our algorithm converges. The algorithm will converge to a local minima since the first term in the objective is smooth and the second term is separable over w. In practice, we observe that it is much more efficient to apply our CD heuristic tofi nd good solutions when λ_2 is small as opposed to the optimization problem to optimality for small values of ϵ using our cutting plane algorithm.

C Real World Case Studies

Here, we present two case studies to showcase the usefulness of MOSS on real-world examples.

C.1 Case Study: Stability in Scientific Discovery. We apply MOSS to a real-world problem: drawing scientific conclusions from observational studies in ecology. Stability (replicability between repeated analyses) is critical to reliable scientific discovery. In Gagné et al. 2018, researchers from the Monterey Bay Aquarium, US Fish and Wildlife Service, and multiple universities sample seabird feathers from multiple locations in the North Atlantic, North Pacific, and South Pacific oceans [15]. They analyze amino acid compounds in each sample to determine the estimated trophic position of the seabird, i.e. the relative position of the animal in its food chain. Organism with higher trophic levels consume organisms with lower ones. The researchers assemble a dataset of 18000 samples and 11 covariates that capture information about the seabird and its surrounding ecosystem. The goal of their study is to understand how these features contribute to trophic position.

To accomplish this, the researchersfi t a random forest model to predict trophic position; the model performs well, with an out-of-sample R^2 of **0.58**. The authors examine the feature importance rankings of the model, determined via in-built variable importance scores [7], and conduct a sensitivity analysis to assess the stability of these rankings by perturbing and re-analyzing the data. Bird species and humanfi shing pressure (catch per unit area) are consistently identified as important drivers of trophic position. This serves as the starting point for our application of MOSS.

Our goal is to construct a sparse subset of decision rules thatfit the data well and that are stable across re-analyses. These decision rules will provide us with a more granular understanding of how the covariates impact trophic position, compared to feature importance rankings. When we apply SIRUS to construct 10 decision rules, we obtain an out-of-sample R^2 of **0.30**, nearly half that of the

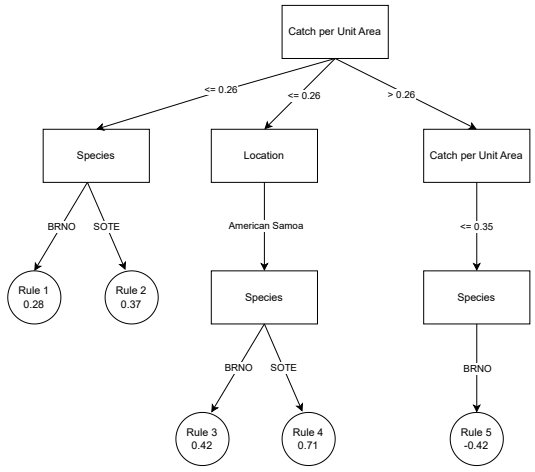

Figure 8: Stable structure of 5 decision rules consistently constructed by MOSS across analyses for the seabird case study.

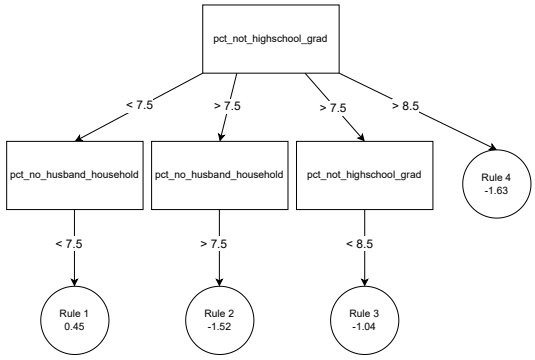

Figure 9: Stable structure of 4 decision rules consistently constructed by MOSS across analyses for the census planning case study.

original random forest. This rule set is not accurate enough to generate reliable conclusions. When we apply FIRE, we obtain an out-of-sample R^2 of **0.59**, however, **0** rules are consistently constructed across repeated analyses. These results are not stable, or replicable, enough to be considered trustworthy. When we apply MOSS, we obtain an out-of-sample R^2 of **0.57**, comparable to that of the random forest and we can identify a structure of **5** decision rules are consistently constructed across re-analyses. We show this structure in Figure 8.

By examining this structure, we can determine that a moderate increase infi shing pressure (catch per unit area) decreases the trophic position of the brown noody (BRNO), that seabirds in America Samoa have higher trophic positions, and the the sooty tern (SOTE) generally has higher trophic positions than the brown noody. Thesefi ndings are more granular than the feature importance rankings presented in [15], and are stable across repeated analyses.

C.2 Case Study: Census Planning. We apply MOSS to the real-world policy problem of identifying how demographic features at the census tract level contribute to low response rates for the American Community Survey (ACS). The ACS is administered annually by the U.S. Census Bureau and collects detailed demographic information on a sample of the U.S. population. ACS data is frequently used to inform policy decisions, including environmental justice and climate action initiatives [20]. As such, ensuring that the data is collected from a representative sample of the population is important for policymaking.

We use data from the U.S. Census Bureau Census Planning Database [29] to predict response rates for the 2017 ACS in California. Our dataset contains demographic information at the census tract level and consists of 8000 observations and 3000 features. Motivated by the observation that tree ensembles have historically performed well in predicting census return rates [12], wefi rstfi t a random forest model. This black-box ensemble performs well, achieving an out-of-sample R^2 score of **0.79**, but lacks transparency.

To improve transparency, we construct a compact collection of decision rules. FIRE generates rule sets of 15 rules that achieve an out-of-sample R^2 score of **0.75**; however, these rules are highly unstable across repeated analyses, making them unreliable. To enhance stability, we apply SIRUS to construct rule sets; but this leads to a decline in predictive performance, reducing the out-of-sample R^2 score to **0.52**.

When we apply MOSS to construct a compact rule set, we achieve an out-of-sample R^2 score of **0.73**, significantly higher than that of SIRUS. Additionally, we identify a set of four decision rules that are consistently reproduced across re-analyses, which we show in Figure 9. From this structure, we observe that high school educational attainment and the percentage of households with no husband present are consistently identified as key drivers of low response rates. Thesefi ndings may help inform strategies to improve response rates for the ACS.

D Experiment Details

In this section, we discuss additional details regarding our experiments.

D.1 Datasets Used. We show a full table of the datasets used in our experiment in §4.2, along with the size of each dataset. All of these datasets were sourced from the OpenML repository [30].

OpenML ID	Name	Observations	Features
296	Ailerons	13750	40
1027	ESL	488	4
42570	mercedes	4209	376
41021	Moneyball	1232	14
183	abalone	4177	8
196	autoMpg	398	7
195	auto_price	159	15
558	bank32nh	8192	32
560	bodyfat	252	14
227	cpu_small	8192	12
216	elevators	16599	18
574	house_16H	22784	16
537	houses	20640	8
189	kin8nm	8192	8
405	mtp	4450	202
344	mv	40768	10
547	no2	500	7
201	pol	15000	48
529	pollen	3848	4
308	puma32H	8192	32
294	satellite_image	6435	36
541	socmob	1156	5
507	space_ga	3107	6
223	stock	950	9
505	tecator	240	124
315	us_crime	1994	127
519	vinnie	380	2
690	visualizing_galaxy	323	4
503	wind	6574	14
287	wine_quality	6497	11

D.2 Experimental Details. We conduct all of our performance experiments on a 2022 MacBook Pro. We implement MOSS in Python, and we will open-source our implementation after the review period. We will also open-source the code to reproduce all of our experiments. For our competing algorithms, we use the following implementations.

- FIRE: We use the open-source Python implementation of FIRE found in the GitHub repository linked in [21].
- GLRM: This method is implemented in the AIX360 package maintained by IBM [2]
- RuleFit: We implement the RuleFit algorithm in SCIKIT-LEARN [8].
- We implement the SIRUS algorithm found in [3] in Python.