Universiteit Leiden
The Netherlands

Opleiding Informatica & Economie

Exploring the Rashomon set using rule lists

Luca van Keulen

Supervisors:
Francesco Bariatti & Matthijs van Leeuwen

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/07/2025

Abstract

We use SamRuLe's sampling bound with the exact CORELS solver to mine thousands of near-optimal, fully interpretable rule lists on five benchmark data sets. The found rule lists let us quantify how quickly each dataset's Rashomon set expands and how its growth is driven by rule-length and feature-usage patterns. These observations provide insights into the datasets' complexity and the interpretability of their models.

Contents

1 Introduction

Over the past decade, interest in *interpretable* machine-learning models has surged. Such models can be inspected directly by domain experts instead of being almost impossible to trace back. Among them, *rule lists* (ordered sequences of human-readable *if-then* statements) have become indispensable. They are complex enough to capture non-linear class boundaries, but simple enough to be explained in a few lines of text. Rule lists enable decision aids in e.g. medicine, and credit scoring & compliance to be transparency or trustworthy. [Riv87, Rud19]
Learning high-quality rule lists is computationally hard. Exact solvers guarantee optimality but slow down when a dataset has many features or instances [ALSA+18]. Heuristic approaches like RIPPER scale better, but do not guarantee that the discovered model is anywhere near optimal.[Coh95]
Pellegrina et al. recently introduced *SamRuLe*, a sampling formula that selects just enough training examples to let an exact approach, such as CORELS, find a near-optimal rule list with high confidence [PV24].
Breiman's *Rashomon effect* states that real-world data often allows multiple models whose predictive performance is almost the same [Bre01]. The set of all such models (those lying within a tolerance of the best) is called the *Rashomon set*. Understanding the *Rashomon set* of all rule lists can reveal which features are important or if it is worth to keep searching for a better rule list. The sampling guarantee of SamRuLe now makes it feasible to systematically explore a lot of near-optimal rule lists. Which opens the door to new research questions:

1. *Can the variability in high-performing rule lists obtained using SamRuLe be used to characterise the Rashomon set?*

2. *What does that characterisation reveal about each dataset's modelling complexity and the characteristics of its near-optimal rule lists?*

This thesis makes the following contributions:

1. We design a pipeline that couples SamRuLe sampling with the CORELS solver to retrieve thousands of near-optimal rule lists across five public data sets of varying size and class balance.

2. We provide a cross-dataset analysis of the datasets' Rashomon sets.

3. We analyse the structure of the rule lists found.

4. All code will be available on a public github page.[1]

[1] https://github.com/Lucavk/Bachelor-Thesis

2 Preliminaries and Background

This chapter introduces the foundational concepts that support the rest of this thesis. In particular, it presents *rule-based classification* and the associated performance metrics, which are crucial for evaluating the rule lists used here. Secondly, the chapter provides the formal definition of the *Rashomon set*, a concept used in later analyses to compare near-optimal rule lists. Finally, it outlines the rule list learning approach adopted in this work: draw a sample with SamRuLe and then apply CORELS to find a near-optimal rule list.

2.1 Rule-Based Classification

Classification is a fundamental machine learning task where we aim to categorise instances (data points) into predefined classes based on their features. For each instance, the classifier independently decides which category it belongs to. In binary classification, which is our focus in this thesis, instances are assigned to one of two classes, specifically positive or negative. Various methods exist for solving classification problems, ranging from statistical approaches to neural networks. In this thesis, we focus on rule-based classifiers.

Rule-based classifiers are a family of models that make their predictions through IF-THEN statements. Each statement, called a rule, checks if the instance satisfies a condition such as *"age > 35"* or *"occupation = technician"*. If the condition is true, the rule immediately assigns the input to a class. For example "will repay the loan" or "is poisonous". Since input features are often human-readable, this type of models is good for interpretability. It is always possible for a human to look back and find out why the model made a certain decision.

A single rule rarely covers the full variety inside the data sets we encounter in real data. Therefore, rule-based systems chain several rules into an ordered list. The prediction is made in a top-down manner. The model tests the first rule; if its condition matches, the corresponding label is returned, and the process stops. Otherwise the second rule is tried. This happens until either a match is found or a final default rule is reached that catches all remaining cases. This makes the order of the rules important. Earlier rules have priority, so we can steer the behaviour of the rule list by placing highly reliable rules at the top and more edge case rules further down.[Riv87]
Although it is possible to use rule lists for multi classification problems, in this thesis we only focus on tasks that have two possible outcomes.

2.1.1 Performance of Rule Lists

Rule lists can vary in how well they predict. To compare different rule lists and understand their strengths and weaknesses, it is important to use a set of quantitative metrics. These metrics allow us to assess not just overall accuracy, but also how well a rule list handles certain aspects of its classifications. Below are some of the most common measures used to evaluate the performance of rule-based classifiers:

- **Accuracy**. The simplest score: the ratio of all correct predictions to the total number of cases. It answers the question "How often is the model right?"

- **Precision** (positive predictive value). Out of all the instances the model marked as positive, precision tells what fraction were truly positive. High precision means few false positives.
- **Recall** (sensitivity, true-positive rate). Of all the truly positive instances in the data, recall shows what portion the model managed to catch. High recall means few missed positives.
- **Specificity** (true-negative rate). The mirror image of recall: of all truly negative cases, it indicates the share that the model correctly rejected. High specificity means few missed negatives.
- **F_1-score**. The harmonic mean of precision and recall, $F_1 = 2\,\frac{\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$. It balances the trade-off between catching positives and avoiding false alarms, especially useful on imbalanced data.

Different errors matter in different applications. A medical test that misses a disease (low recall) is far more serious than one that raises an occasional false alarm (moderate precision). A spam filter, for example, prefers the opposite balance. Reporting several complementary performance measures paints a fuller picture of the models behaviour and protects against misleadingly high accuracy on imbalanced data sets. [SL09]

2.2 Rashomon Sets

Machine-learning applications typically search for a single "best" model. For instance, the rule list with the highest accuracy. However, in 2001, Breiman observed that real data usually allow many different models whose performance is almost the same. Breiman called this the Rashomon effect.[Bre01]
Using the idea of the Rashomon effect in machine learning, we can explore the variety of models that fit a given dataset well. The Rashomon set is the collection of all models that perform nearly as well as the best one, within a small tolerance(θ). This set can help us understand in how many different ways we can explain the same data. We can define the Rashomon set for a given hypothesis class $\mathcal{F}$ (for example, rule lists with at most 5 conditions), a loss function $\hat{P}$ (for example, accuracy), and a tolerance parameter θ as follows: [SR19]

$$\hat{R}_{\text{set}}(\mathcal{F},\theta) := \left\{ f \in \mathcal{F} \;:\; \hat{P}(f) \le \hat{P}(\hat{f}) + \theta \right\}, \quad \text{where} \quad \hat{f} \in \arg\min_{f\in\mathcal{F}} \hat{P}(f)$$

When we apply this Rashomon framework specifically to rule lists, it becomes particularly valuable because rule lists aim to be human-interpretable. Instead of focusing on a single "best" rule list, we can systematically explore and compare multiple rule-based explanations that perform nearly as well, but some could be more interpretable than others. This could help us to understand the structure of near-optimal rule-based models.

2.3 Strategies for Learning Rule Lists

A *rule list* is an ordered sequence of **if–then** clauses followed by a default rule (**else**). Selecting the rules and their order is combinatorial, so the finding problem could become difficult. Two approaches to finding good-performing rule lists are mainly used:

- **Heuristic search:** Greedy learners such as RIPPER[Coh95] add, delete, or tweak one rule at a time. This technique makes them fast and allows them to scale to very large data sets. The downside is that they do not guarantee that the final model is close to the best model possible. Small changes in the data can lead to very different rule lists.
- **Exact search:** Exact algorithms explore the entire space. They consider all possible rule lists and return the one with the best performance. This approach guarantees that the returned model is optimal, but it can be very slow. Exact methods are often impractical for large data sets or many features. The exact approach that we use in this thesis is CORELS (Certifiably Optimal RulE ListS)[ALSA+18]. CORELS provides strong optimality guarantees while using efficient search strategies: branch-and-bound techniques (which eliminate unpromising paths early) and symmetry-aware pruning (which avoids redundant evaluations of equivalent rule orderings) to search the space much faster than previous exact methods. It minimises $\hat{R}(L) + \lambda \operatorname{len}(L)$, where $\hat{R}(L)$ is empirical loss and λ is a small penalty for larger rule lists.

2.3.1 SamRuLe

On very large data sets, CORELS can be slow. A recent study by Pellegrina et al. [PV24] introduces a sampling-based approach to rule list learning called SamRuLe. It is not a search algorithm, but it uses a formula to determine how many instances we need to find a near-optimal rule list. The study showed that it is possible to sample a small subset of the data, run an exact solver (CORELS) on that subset, and still find a rule list that is close to the best one possible. They created a formula that tells how many instances m we must keep so that CORELS on that subset returns a rule list whose performance is near-optimal.
A rule list with *length* k has at most k rules, and each rule may contain up to z combinational features. The larger k or z becomes, the more rule list options there are. SamRuLe measures this complexity with the *VC-dimension* and shows that it grows on the order of

$$\operatorname{VCdim}\Big(\text{rule lists}\Big) = \mathcal{O}\Big(k\, z\, \log(d/z)\Big),$$

where d is the number of binary features in the data set.
Using that VC-dimension bound, the authors express the minimum sample size m as a function of the six variables listed in Table 1. In short, stricter accuracy ($\downarrow \varepsilon$), higher confidence ($\downarrow \delta$), or a larger hypothesis space (higher k, z, or d) all push m upward, whereas the stabiliser θ prevents m from exploding when the optimal rule list makes almost no errors.
With a sample of exactly m rows, SamRuLe guarantees that the rule list $\tilde{L}$ found by CORELS satisfies, with probability at least $1 - \delta$,

$$\hat{R}(\tilde{L}) \leq \hat{R}(L^\star) + \varepsilon \max\{\hat{R}(L^\star), \theta\},$$

where $L^\star$ is the best rule list on the full data set.

Any sample of the size m is large enough to learn a rule list that is almost as good as the optimal one. Because the same guarantee holds for every new sample of size m, we can draw several such samples and rerun the exact solver each time. This produces a set of different rule lists, all backed by the same near-optimal promise.

Variable	**Description**	**Effect on m**
m	Minimum sample size required to find a near-optimal rule list	–
k	Maximum length of rule list (number of rules)	$\uparrow k \rightarrow \uparrow m$
z	Maximum number of combinational features in each rule	$\uparrow z \rightarrow \uparrow m$
ε	Accuracy tolerance (how close to optimal)	$\downarrow \varepsilon \rightarrow \uparrow m$
θ	Small constant that keeps error bound reasonable when optimal rule list has few errors	$\downarrow \theta \rightarrow \uparrow m$
δ	Allowed failure probability (e.g., $\delta = 0.05$ means 5% chance the guarantee fails)	$\downarrow \delta \rightarrow \uparrow m$
d	Number of binary features in the dataset	$\uparrow d \rightarrow \uparrow m$

Table 1: Variables used to calculate minimum sample size in SamRuLe

3 Approach and Methodology

To answer the research question, we propose a systematic approach that combines data selection, sampling and feature selection, and rule list mining using SamRuLe and CORELS. This section outlines the methodology used to achieve this goal.
Our methodology follows four main steps. First, we select five diverse datasets with varying characteristics to provide a diverse benchmark. Second, we reduce dataset size through sampling and feature selection to ensure computational feasibility. Third, we implement an experimental design using 5 fold cross-validation and sampling with SamRuLe. Finally, we generate rule lists using CORELS and evaluate their performance across multiple metrics.
This structured approach allows us to systematically explore rule lists across different datasets while maintaining reasonable computation times, with both experimental setups completing within 15 hours per dataset.

3.1 Data Collection and Preparation

3.1.1 Data Selection

To be able to compare observations about the rashomon set between different data sets, we select five datasets: *Adult*, *Bank*, *IJCNN1*, *Mushroom* and *SUSY*, from the seven that Pellegrina et al. used to their research about SamRule [PV24]. We choose those five data sets to get a good variety of data sets that differ in size, class balance, and feature size.*Adult* and *Bank* are medium-sized (around 35 000 instances) with a large class skew; *IJCNN1* adds a larger instance count and an even more extreme imbalance (90% majority class); *Mushroom* provides an almost balanced scenario; and *SUSY* contributes an extremely large dataset. Their characteristics are summarised in Table 2.

Table 2: Characteristics of the original datasets

Dataset	Total Samples	Features	Balance	Majority Class (%)
Adult	32 561	199	Imbalanced	75.9%
Bank	41 188	155	Highly imbalanced	88.7%
IJCNN1	91 701	34	Highly imbalanced	90.5%
Mushroom	8 124	124	Balanced	51.8%
SUSY	5 000 000	139	Balanced	56.0%

The CORELS algorithm requires that all features are binary[ALSA+18]. Fortunately, all five datasets were already converted to binary. The original SamRuLe authors had previously handled the necessary preprocessing by converting continuous variables (e.g. age and income in *Adult*) into supervised bins and applying one-hot encoding to categorical variables [PV24].

3.1.2 Feature and Instance Selection for Scalability

In its raw the datasets contain up to 199 features and up to 5 million instances. Running either SamRule or CORELS on such large data sets would exceed both the 16 GB RAM ceiling of our workstation and reasonable time limitations. To fix this issue we perform a two-stage data-thinning

step that filters rows and columns.

The first stage consists of filtering the rows. Among our datasets, only SUSY requires this step due to its enormous size (5 million instances). We reduce SUSY to 100,000 instances using stratified sampling, which allows us to control the class distribution during sampling. While stratified sampling can preserve the original distribution, for this particular dataset we intentionally adjust the original 56/44 class balance to a perfect 50/50 ratio. This adjustment is feasible due to the large number of available instances and provides us with a perfectly balanced dataset for analysis.

The next step is feature dimensionality reduction. To ensure computational efficiency and model performance, we implement a feature selection process based on a 100-tree Random Forest importance scoring. This approach evaluates each feature's predictive power across the dataset and selects those with the highest information gain. We reduce all datasets except IJCNN1 to 25 features, retaining only the most informative attributes that contribute significantly to class prediction. For IJCNN1, which originally has only 34 features, we keep all features since this modest dimensionality already provides reasonable computational performance without requiring further reduction.

After completing these preprocessing steps Table 3 presents the final characteristics of the five datasets.

Table 3: Characteristics of the experimental datasets

Dataset	**Total Samples**	**Features**	**Balance**	**Majority Class (%)**
Adult	32 561	25	Imbalanced	75.9%
Bank	41 188	25	Highly imbalanced	88.7%
IJCNN1	91 701	34	Highly imbalanced	90.5%
Mushroom	8 124	25	Balanced	51.8%
SUSY	100 000	25	Perfectly balanced	50.0%

3.2 Experimental Design and Rule-List Mining

The goal is to uncover, for each of the five chosen binary-classification datasets, a rich collection of rule lists that are near-optimal in terms of accuracy compared to the best possible list. This collection of rule lists could give us insight into the characteristics of the Rashomon set and dataset. At the same time the entire pipeline has to finish within 15 hours per dataset on a normal workstation, so every design choice balances exhaustiveness against runtime.

Each dataset splits into five stratified folds with an 80:20 train-test ratio. Stratification preserves the original class balance, so that each fold is a fair representation of the whole dataset. Using five folds exposes how sensitive the Rashomon statistics are to natural sampling variation while keeping runtime reasonable. What happens for every fold is shown in Algorithm 1 and explained below.

Next we define the search space. The rule combination depth is fixed at $z = 1$ (one condition per rule), while the maximum list length k either stays fixed at 13 or loops through $2, \ldots, 13$. Those

two settings define our *two experimental setups*. In the constant-maximum-rule-list-length run we repeat k=13 exactly $R = 2000$ times per fold; in the variable-maximum-rule-list-length run we repeat each k-value $R = 80$ times. Both experimental setups use seeds to ensure reproducibility, and each run is independent of the others.

Before training, we use the SamRuLe bound to find the minimal sample size m with accuracy tolerance $\varepsilon = 0.025$, stabiliser $\theta = 0.01$, and failure probability $\delta = 0.05$. We choose these variables because the authors of SamRuLe used the same values in one of their experiments. Drawing exactly m instances guarantees, with 95 % confidence (since $\delta = 0.05$), that the learned rule list stays within 2.5 % of the unknown optimum, yet trims runtime so the 15-hour cap is never exceeded.
Drawing exactly m instances guarantees, with 95 % confidence, that the learned rule list stays within 2.5 % of the unknown optimum, yet trims runtime so the 15-hour cap is never exceeded.

The sampled subset goes into CORELS with a tiny ℓ_0 penalty $\lambda = 10^{-4}$. We use this value because it was used during the experiment from SamRuLe. CORELS enumerates every rule list up to length k and returns the one minimising $\widehat{R}(L) + \lambda \operatorname{len}(L)$. Additionally we run CORELS once per fold on the entire training split. Finding the exact optimum allows us to directly compare the performance of the sampled rule lists to the true best rule list for each fold.

Finally, each list is evaluated on its corresponding held-out 20 %. Accuracy, precision, recall, F_1, and specificity are calculated using scikit-learn [PVG+11]. Together with the rule list, and the chosen variables, those metrics form the raw material for the analysis.

A post-processing step removes duplicate rule lists before saving. Because CORELS can rediscover the same list from different samples, the number of distinct models is often far smaller than the raw number of runs. We also remove all rule lists with a recall of 0. When the recall is 0, the rule list does not cover any positive instances and always predicts the negative class. Such rule lists are not useful for analysis, and we remove them to avoid distorting the results.
Each dataset now ends up with two collections of results, one for the constant-maximum-rule-list-length setup and one for the variable-maximum-rule-list-length setup, containing only deduplicated results.

Dataset-specific note. The Mushroom dataset contains just 8124 instances. For most $(k, \varepsilon, \theta, \delta,)$ combinations, the SamRuLe bound yields m larger than the available training data. This means that no sample can be drawn for those configurations. For the constant-k experiment, this happens for all runs, and for the variable-maximum-rule-list-length experiment, it happens when the maximum k is larger than 3. We keep the Mushroom dataset in the analysis, but keep the issue in mind in the following sections.

Algorithm 1: Pipeline executed inside a single cross-validation fold

Data: Train set $\mathcal{D}_{\text{train}}$; test set $\mathcal{D}_{\text{test}}$
Result: Two collections $\mathcal{R}_{\text{const}}$ and $\mathcal{R}_{\text{var}}$ with unique rule lists and metrics
for $r = 1$ **to** 2000 **do** `// constant-maximum-rule-list-length experiment`
    $m \leftarrow (13, 1, \varepsilon, \theta, \delta, d)$;
    $\mathcal{S} \leftarrow$ sample m points from $\mathcal{D}_{\text{train}}$;
    $L^\star \leftarrow \text{CORELS}(\mathcal{S}, 13, \lambda)$;
    Evaluate $L^\star$ on $\mathcal{D}_{\text{test}}$ and append record to $\mathcal{R}_{\text{const}}$;
for $k = 2$ **to** 13 **do** `// variable-maximum-rule-list-length experiment`
    for $r = 1$ **to** 80 **do**
        $m \leftarrow (k, 1, \varepsilon, \theta, \delta, d)$;
        $\mathcal{S} \leftarrow$ sample m points from $\mathcal{D}_{\text{train}}$;
        $L^\star \leftarrow \text{CORELS}(\mathcal{S}, k, \lambda)$;
        Evaluate $L^\star$ on $\mathcal{D}_{\text{test}}$ and append record to $\mathcal{R}_{\text{var}}$;
$\mathcal{R}_{\text{const}} \leftarrow \text{removeDuplicates}(\mathcal{R}_{\text{const}})$;
$\mathcal{R}_{\text{var}} \leftarrow \text{removeDuplicates}(\mathcal{R}_{\text{var}})$;

4 Results and Analysis

This section presents our findings on characterising Rashomon sets for rule lists across five datasets. Our analysis progresses through three main areas. First, we analyse the size and shape of Rashomon sets across different performance metrics. Next, we analyse structural patterns in the discovered rule lists, including the number of unique rule lists, feature usage and positional importance. Finally, we investigate the relationship between rule list complexity and performance, identifying the minimum complexity needed for effective prediction.

4.1 Rashomon Set Findings

We begin our analysis by characterising the Rashomon sets, which are all rule lists that sit within a tolerance θ of the best one, using experimental setup 1. In that setup, every model is allowed to have up to thirteen rules ($k_{\max} = 13$). The maximum complexity of these rule lists is fixed, so we can focus on how the Rashomon set size changes at different thresholds for different folds, datasets, and performance measures.
The analysis consists of two parts: we first ensure that the cross-validation folds behave similarly, and then compare the five datasets using five different metrics (accuracy, precision, recall, specificity and F_1).

4.1.1 Rashomon Set Sizes Across Folds

Before comparing the datasets, we confirm that the Rashomon sets are *consistent across folds*. Fold-to-fold consistency matters because it supports the assumption that any observed dataset-level differences are not the result of a particular subset.
Figure 1 and 2 introduce our Rashomon set size plots. The horizontal axis sweeps the performance tolerance θ from 0 to 0.20, while the vertical axis shows the percentage of unique rule lists whose score lies within that tolerance of the best model in the same fold. Each coloured line represents a single cross-validation fold and is annotated with the absolute number of rule lists uncovered in that fold (after deduplication). Because we normalise by the fold's own total, the curves can be compared directly even though each fold produced a slightly different count of candidate models. At $\theta = 0$ only the optimal rule list sits in the Rashomon set, so every curve starts at 0%. As θ increases, we let more error in, so the set grows. The steeper the line, the closer the other 'near optimal' rule lists are to the optimal rule list.
For the `ijcnn1` dataset the five accuracy curves in Figure 1 overlap almost perfectly, and the same pattern holds for the F_1 curves in Figure 2. In other words, changing the train/test split hardly changes how many rule lists qualify at any threshold. This fold-to-fold consistency tells us that duplicate rule lists discovered under different random seeds do not distort the picture after de-duplication. It also tells dataset-level differences are not a result of a specific fold, which makes it possible to compare multiple datasets. We observe the same near-identical curves for every other dataset and performance metric. These plots are collected in Appendix A.1.

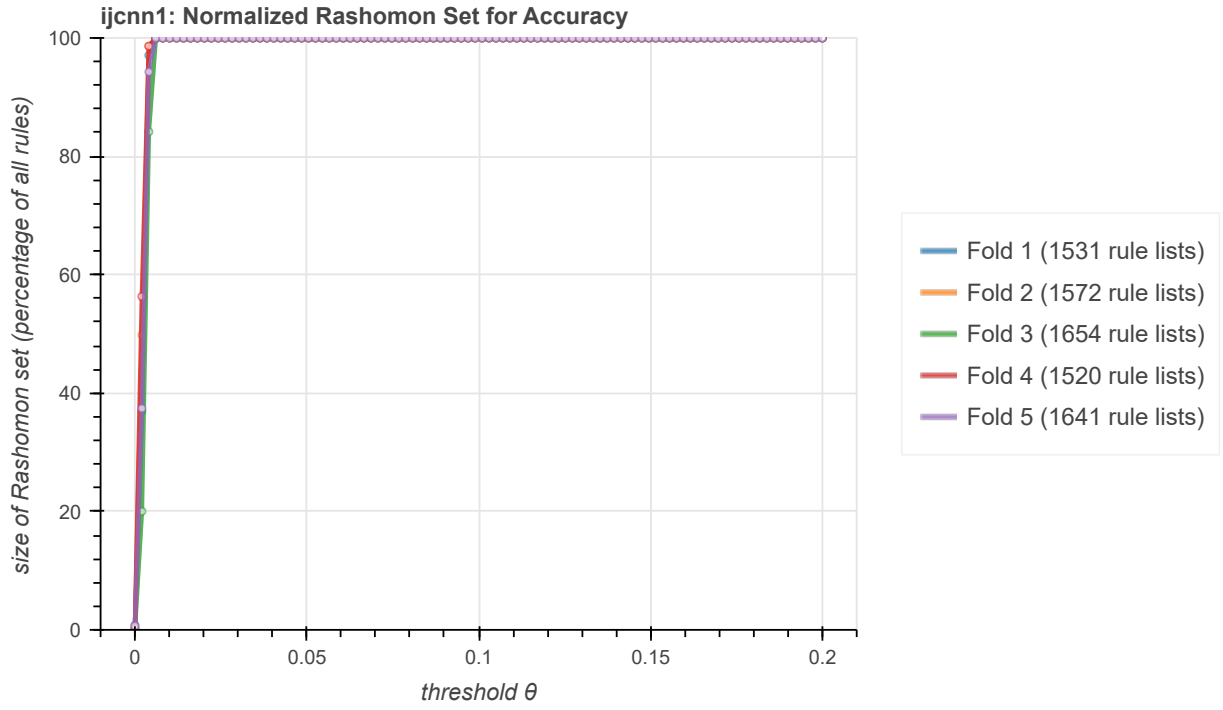


Figure 1: Share of rule lists in the Rashomon set at thresholds between 0 and 0.2 for every fold of `ijcnn1` (accuracy).

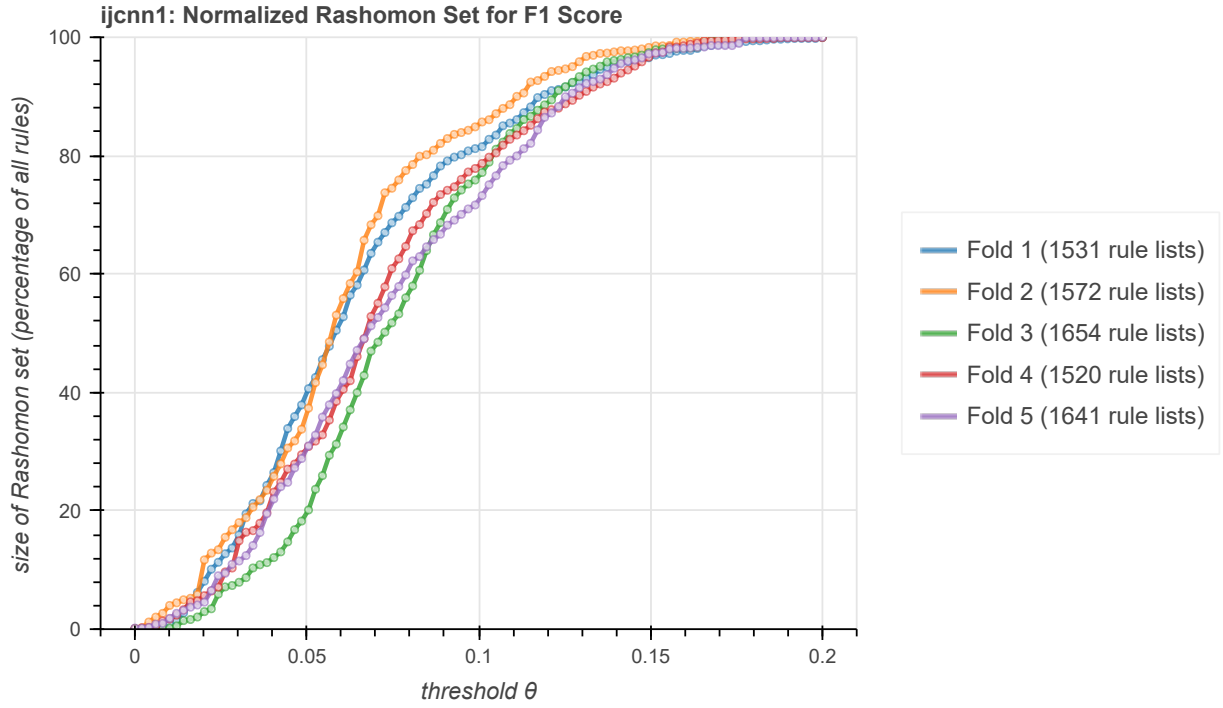


Figure 2: Share of rule lists in the Rashomon set at thresholds between 0 and 0.2 for every fold of `ijcnn1` (F1 Score).

4.1.2 Rashomon Set Sizes Across Datasets

The figures, 3-4, in this section, compare the five datasets on five different metrics. Every plot shows the normalised Rashomon set size, the share of unique rule lists whose score is within a tolerance θ of the best list in the same dataset, on the vertical axis and the tolerance $\theta \in [0, 0.20]$ on the horizontal axis. The coloured planes span the minimum-maximum across folds, so the thickness of the plane is a quick visual cue for fold variation (thin means the folds are similar, thick means they are less similar). Because the curves are normalised by each dataset's own model count, differences in absolute search inventory do not impact the comparison.

Accuracy. Figure 3 shows that every dataset rises extremely fast upward once θ exceeds about 0.02. That behaviour is expected because CORELS is steered by a loss function that mirrors misclassification error. Once we allow even a tiny drop in accuracy, we let in almost all other rule lists. The `mushroom` and `susy` curves rise fastest, hitting approximately 100 % of the models by $\theta \approx 0.004$, whereas `adult` is slightly behind and does not reach 100 % until $\theta \approx 0.012$.

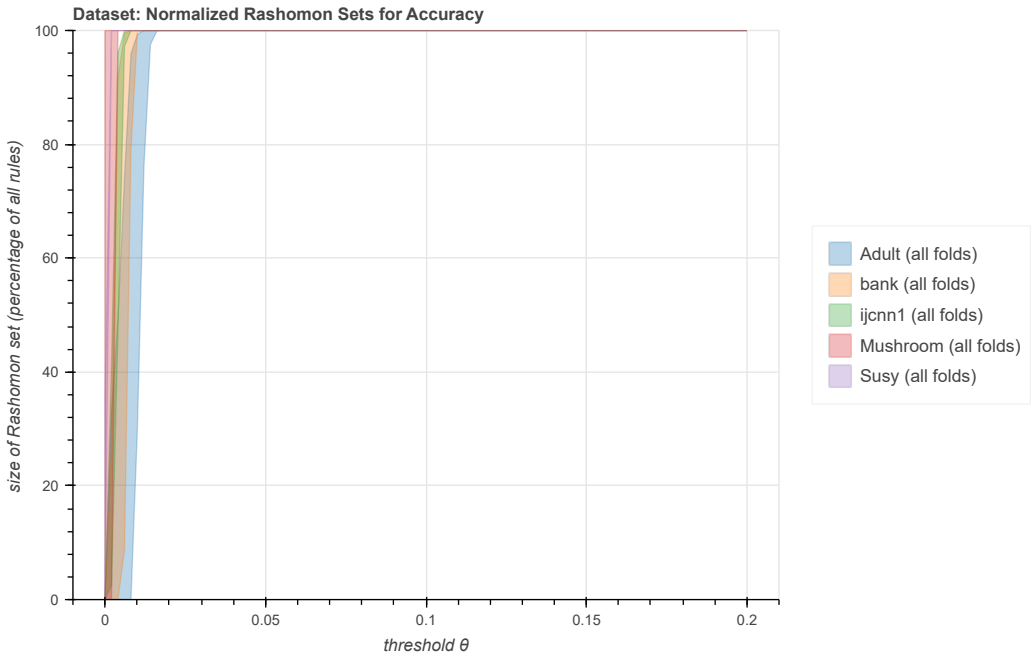


Figure 3: Accuracy

Specificity. Figure 4 tracks the true-negative rate. For `bank`, `ijcnn1`, `mushroom`, and `susy`, the specificity plane rises almost in step with the accuracy plane. Once a rule list is near-optimal in accuracy, it is also near-optimal in the number of negatives it predicts correctly.
however `adult` behaves differently. Many rule lists that sit within a tight accuracy tolerance still vary widely in how many false positives they allow, so the specificity Rashomon set grows more slowly. In practical terms, `adult` contains several near-optimal lists that trade extra false positives for gains elsewhere, producing a broader spread on specificity than on accuracy.

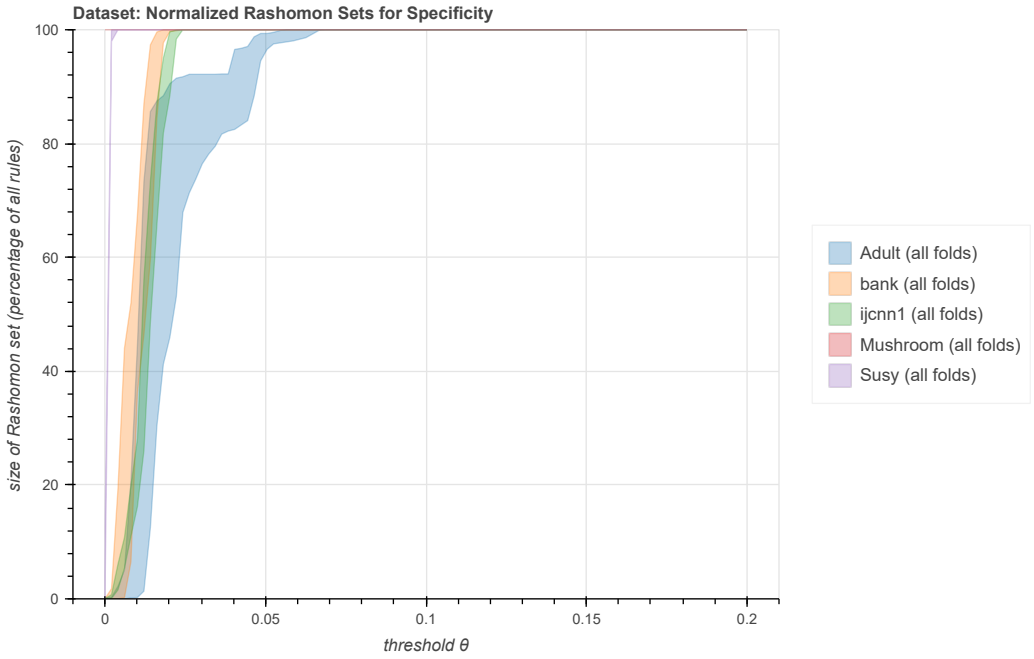


Figure 4: Specificity

Precision and Recall. Figures 5 and 6 split the performance into how strong the model is when it predicts positive (precision) and how often it actually finds the positives (recall). These two figures show the largest differences between datasets. `mushroom` and `susy` again have the steepest curves. For all 4 independent performance measures (not derived like F1) these two datasets reach 100 % of their models before θ exceeds 0.01. This means that the Rashomon set is very large even at very low thresholds. For these two datasets, all rule lists that we found could be used as a useful classifier. This raises the question wether it is possible to use less complex models while preserving the same performance, which will be answered later in Section 4.2.3.

Precision. The other three datasets, `adult`, `bank`, and `ijcnn1`, behave more diversely. Looking at precision, they show 2 different patterns. `Ijcnn1` stands out for its wide spread: up to a tolerance of $\theta = 0.10$ only $\approx$ 9-19% of all rule lists qualify, indicating that very few models match the best-precision model. Beyond that point, the curve climbs steeply and reaches a Rashomon set size of 100% at $\theta \approx 0.19$. `Bank` and `Adult` show a similar delayed rise. Between $\theta = 0.01$ and 0.05 the Rashomon set begins to grow; `Bank` finishes closely afterwards (all models included by $\theta \approx 0.06$), whereas `Adult` levels off briefly and needs until $\theta \approx 0.10$ before the last models join. That late-arriving tail for `Adult` mirrors the late slow rise we observed in its specificity curve (Figure 4).

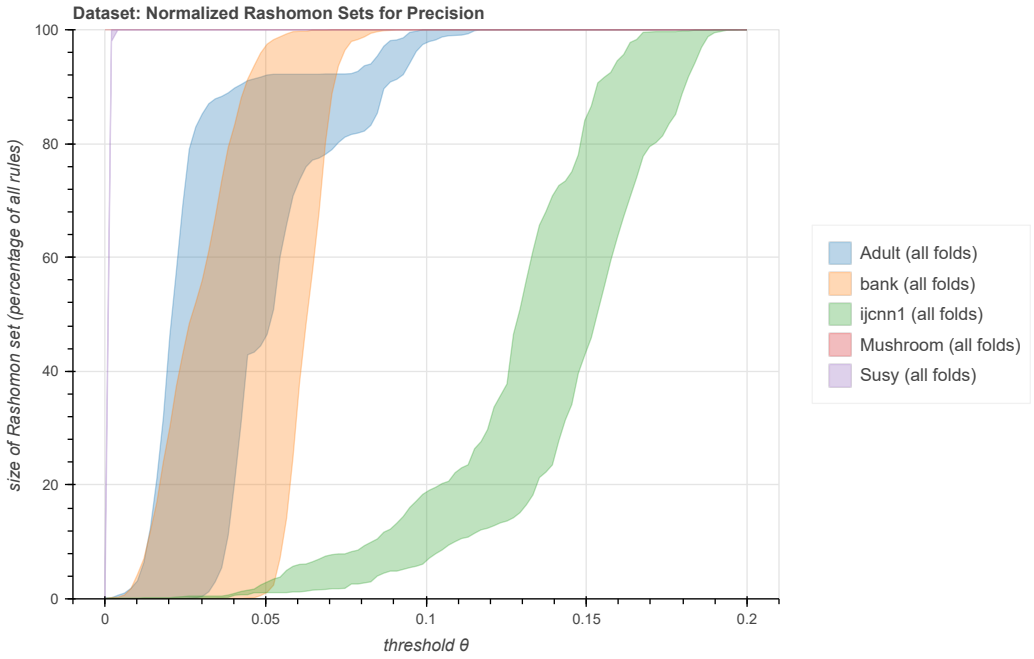


Figure 5: Precision

Recall. The recall curves re-order the story. For `Bank`, the Rashomon set barely grows until $\theta \approx 0.03$; from there it increases steadily and reaches full size by $\theta \approx 0.15$, with a brief pause around 0.06-0.08. `IJCNN1` follows the same shape but is stretched horizontally: even at $\theta = 0.20$ about 3-5% of the rule lists remain outside the set, confirming that a handful of models still miss too many positives. `Adult` rises slowest. At $\theta = 0.10$ only 10-18% of its rule lists are part of the Rashomon set in recall. The remainder do not join until θ passes 0.13, and even at $\theta = 0.20$ a small fraction is still excluded. Taken together, these patterns say that high-recall models are scarce in strongly imbalanced data (Bank, IJCNN1) and even scarcer in `Adult`. The rule lists often trade recall for other gains, which could be explained by the loss function that CORELS uses to select the optimal rule lists.

F_1 Score. Figure 7 combines recall and specificity through their harmonic mean. For `susy` and `mushroom`, the F_1 curve is almost identical to the specificity and recall curve. The Rashomon set is huge even at tiny θ. For `adult`, `bank`, and `ijcnn1`, the F_1 curves lie slightly above their recall curves but keep the same shape, confirming that specificity is not the limiting factor but recall is. As a result F_1 offers the clearest overview of balanced performance across rule lists Datasets whose recall grows slowly (Adult, IJCNN1 and Bank) show a correspondingly slow F_1 expansion, whereas datasets with similar recall scores (Mushroom and SUSY) display a rapidly rising curve. This view underscores how the Rashomon set's shape depends less on accuracy than on the model's willingness to capture the minority class.

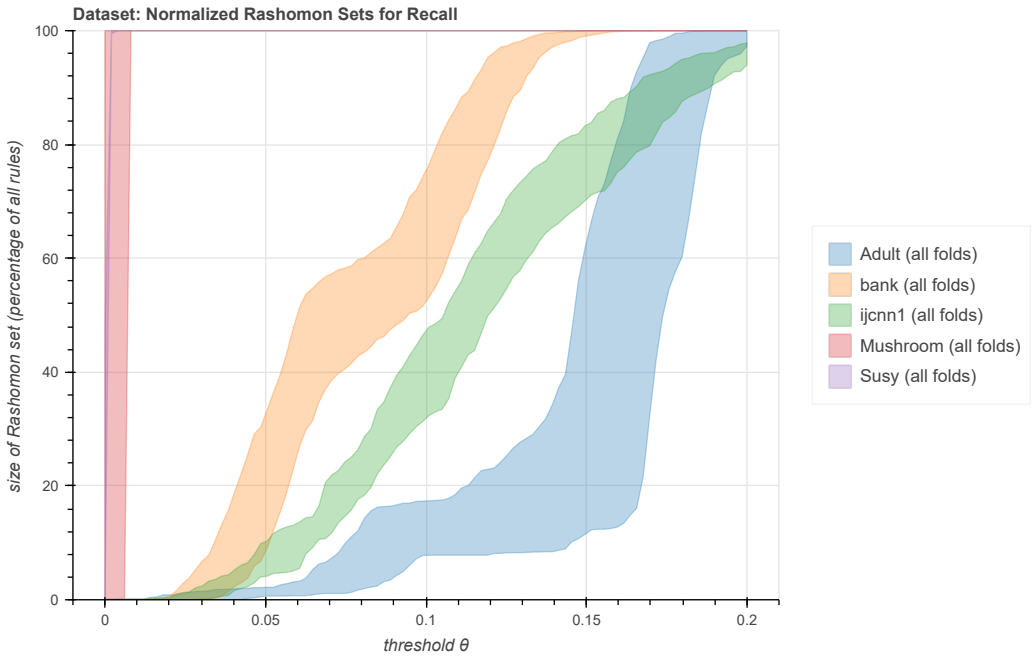


Figure 6: Recall

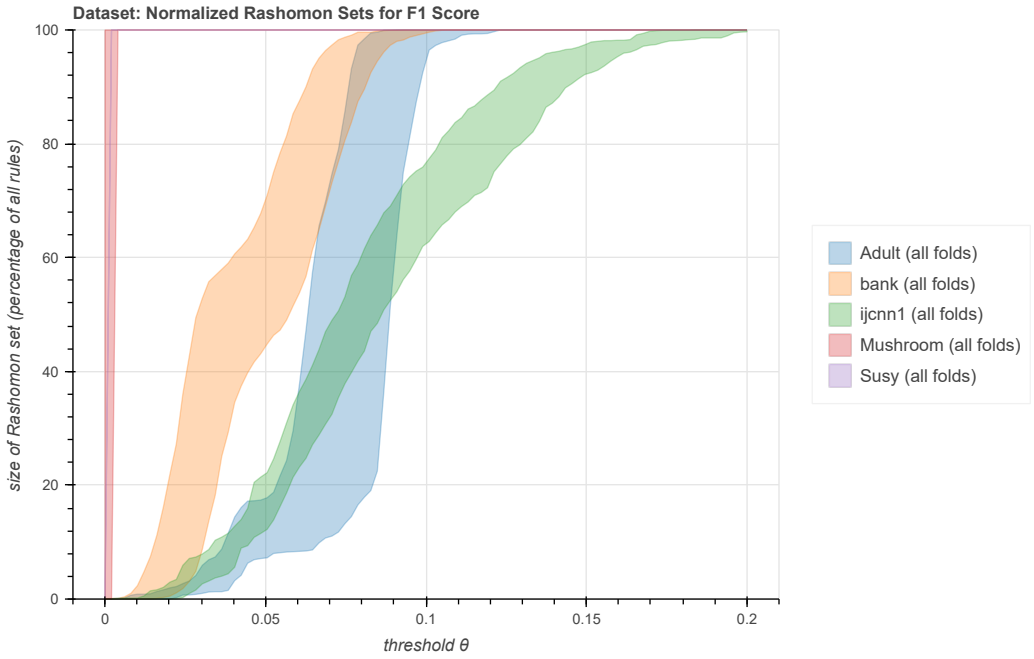


Figure 7: F1 Score

4.2 Rule list structure Discoveries

Having examined the Rashomon sets across performance metrics using Experimental Setup 1, we now introduce our second experimental approach while continuing to leverage results from the first. In Experimental Setup 2 (variable-maximum-rule-list-length), we constrain CORELS to search specifically at each rule list length k $\in$ 2,...,13, allowing us to isolate the effect of rule list length on performance.

Our analysis proceeds through three complementary perspectives: first observing the amount of unique rule lists found across datasets, then analyzing the feature usage patterns within these lists, and finally investigating how rule list complexity relates to performance.

4.2.1 How many different rule lists are found?

In this section, we compare and analyse how many different rule lists are found in both experimental setups, constant-maximum-rule-list-length and variable-maximum-rule-list-length. In Section 4.1, we analysed the Rashomon set sizes for various threshold *theta* values across folds and datasets. The comparison used a normalised percentage of rule lists in the Rashomon set. This means that the number of rule lists found is not taken into account. In this section, we will compare the number of unique rule lists found in both experimental setups.

Constant-maximum-rule-list-length Table 4 presents the number of unique rule lists found in the constant-maximum-rule-list-length experiment for each dataset and fold. This experiment consists of 2000 runs per fold with a maximum rule list length of 13, plus 1 exact run per fold. This means that the maximum unique rule lists found per fold is 2001. Several patterns emerge from these results:

First, we observe that the Mushroom dataset has by far the fewest unique rule lists, with an average of only 15 per fold. This can be explained by the SamRuLe algorithm's minimum sample size requirement. For this dataset, the minimum sample size calculated by SamRuLe (19 891 instances) far exceeds the total number of available training instances (only 6 499). As a result, instead of using different samples for each run as intended, CORELS was forced to use the same complete training data in each run, leading to identical rule lists being found across iterations. Knowing this, we expect to see one unique rule list per fold, since CORELS finds the optimal rule list.

This sampling limitation directly explains the extremely low diversity in the results for Mushroom. In contrast, the Bank dataset consistently produces the highest number of unique rule lists, with an average of 1 771 per fold. This indicates a diverse space of near-optimal rule lists for this dataset and suggests that the Bank dataset contains a variety of feature combinations that can achieve similar predictive performance.

The Adult and IJCNN1 datasets show intermediate behaviour with averages of 1,654 and 1,583 unique rule lists per fold. These values suggest that while there is substantial diversity in the rule lists for these datasets, the space is not as expansive as for Bank.

Interestingly, the SUSY dataset, despite having the largest number of instances, finds the lowest number of unique rule lists among the large datasets (average 1 113 per fold). This may indicate that,

for SUSY, the space of near-optimal rule lists is more restricted, possibly due to a few dominant features or patterns that consistently lead to the best performance.
Across all datasets, the number of unique rule lists found is much lower than the total number of runs (2 001 per fold). This shows the tendency of CORELS to rediscover the same rule lists from different samples.
The diversity of near-optimal rule lists varies substantially across datasets. Section 4.2.2 further explores the features used in these rule lists, providing insights into the underlying patterns that contribute to this diversity.

Table 4: Number of unique rule lists found in constant-maximum-rule-list-length experiment (max length = 13, 2000 runs + 1 exact run per fold)

Dataset	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Adult	1,532	1,732	1,622	1,685	1,699	1,654
Bank	1,738	1,748	1,792	1,781	1,799	1,771
IJCNN1	1,531	1,572	1,654	1,520	1,641	1,583
Mushroom	16	14	16	16	15	15
SUSY	1,075	1,208	981	1,176	1,124	1,113

Variable-maximum-rule-list-length Figure 8 plots, for each data set, the *average* number of CORELS rule lists that are *unique within a fold*. The y-axis counts those unique lists; the x-axis shows the length cap k. For every k each fold was run 80 times, so the theoretical maximum is 80 distinct lists per fold.
As expected, `mushroom` offers only a handful of unique rule lists across all length caps. Once the cap reaches $k = 3$, the minumum sample size already is 7 313, which is more than the entire training set of 6 499 instances. In general, the search space expands exponentially with k: the smaller the maximum k, the fewer unique lists can exist at all.
The count of unique lists for a given k can be interpreted as the number of different ways the data can be described with at most k rules. For `bank`, and to a lesser extent for `ijcnn1` and `adult`, many such descriptions appear even when k is small, signalling multiple alternative patterns. By contrast, the `susy` dataset has much less variety at lower maximum rule list lengths and shows a more gradual increase in unique rule lists as k approaches 13. The same pattern shows again that SUSY's behaviour could be dominated by a few key features or patterns that consistently lead to the best performance, restricting the diversity of effective rule lists.

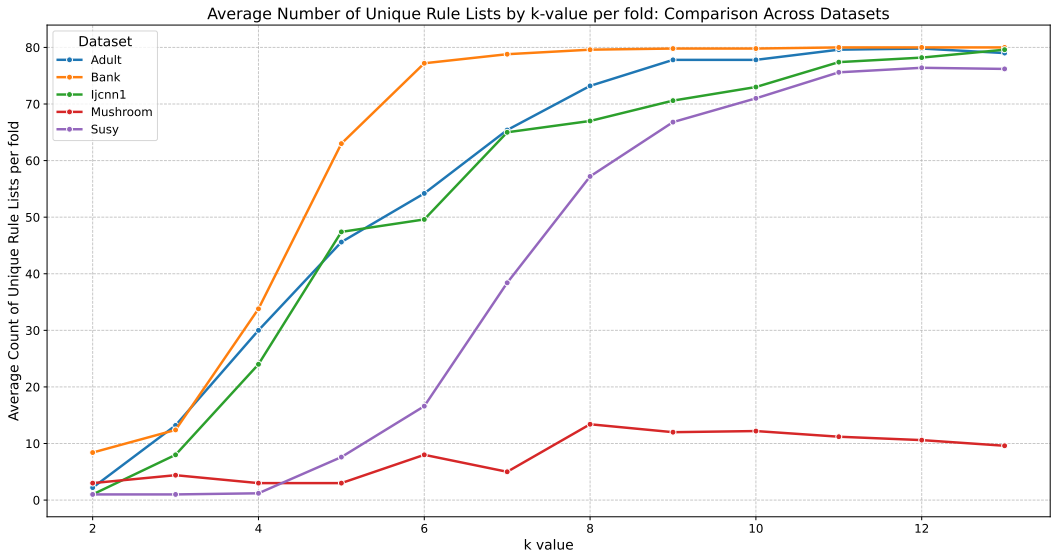


Figure 8: Comparison of datasets based on the average number of unique rule lists found for every k value (rule list length) in the variable-maximum-rule-list-length experiment.

4.2.2 Feature analysis of the rule lists

This subsection dives into the features that appear in the rule lists learned in Experimental Setup 1 (the constant-maximum-rule-list-length experiment) through the lens of their feature usage. Our analysis, therefore, looks at three angles: first, how frequently each feature appears across all lists; second, which particular features co-occur in the same rule list; and third, the position a feature occupies. To keep the analysis concise we compare the two most contrasted cases: `bank`, which produced the largest number of distinct lists (8 858), and `susy`, which produced the fewest outside of `mushroom` (5 564). The other results are available in the Appendix A.2

Feature frequency. Figures 9 and 10 rank the features of each data set by the frequency with which they appear in any rule. In `susy`, nine features appear with roughly equal, very high frequency. After that, the count drops steeply. This indicates that the core of `susy` is tightly packed around those nine features, which are indispensable for constructing near-optimal models. The steep drop-off suggests that most other features contribute little to the overall performance, leading to a limited number of unique rule lists.
In `bank`, the first two variables occur in most rule lists, but unlike `susy`, the count doesn't drop steeply afterwards. Instead, it maintains a steady, gradual decline across many features. This extended plateau indicates that many different features contribute to the rule lists, providing multiple alternative ways to create near-optimal models.

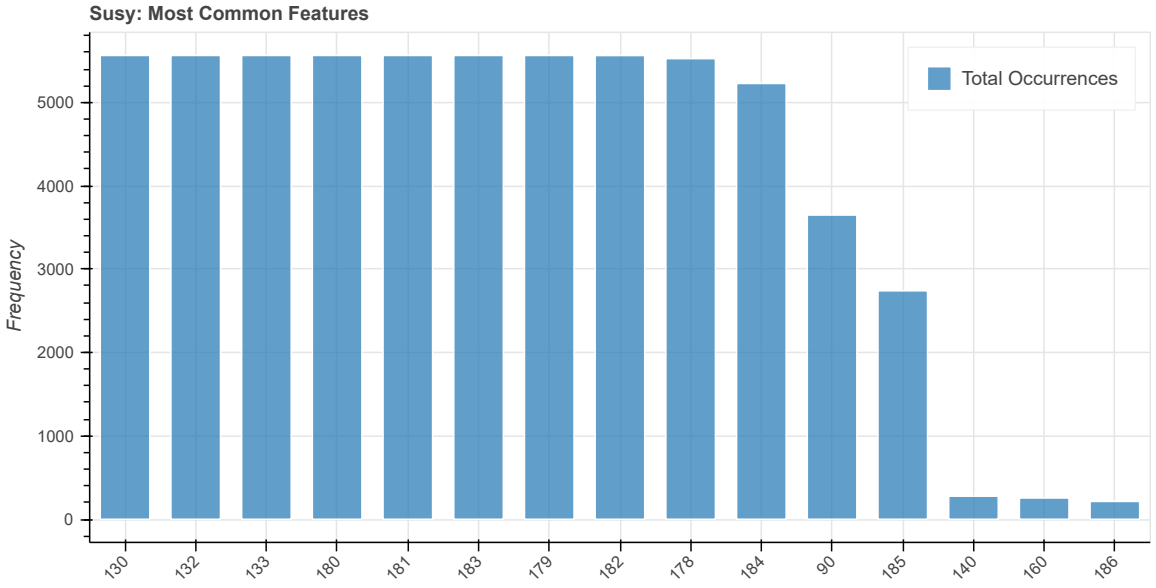


Figure 9: Feature occurrence bar chart for the Susy dataset.

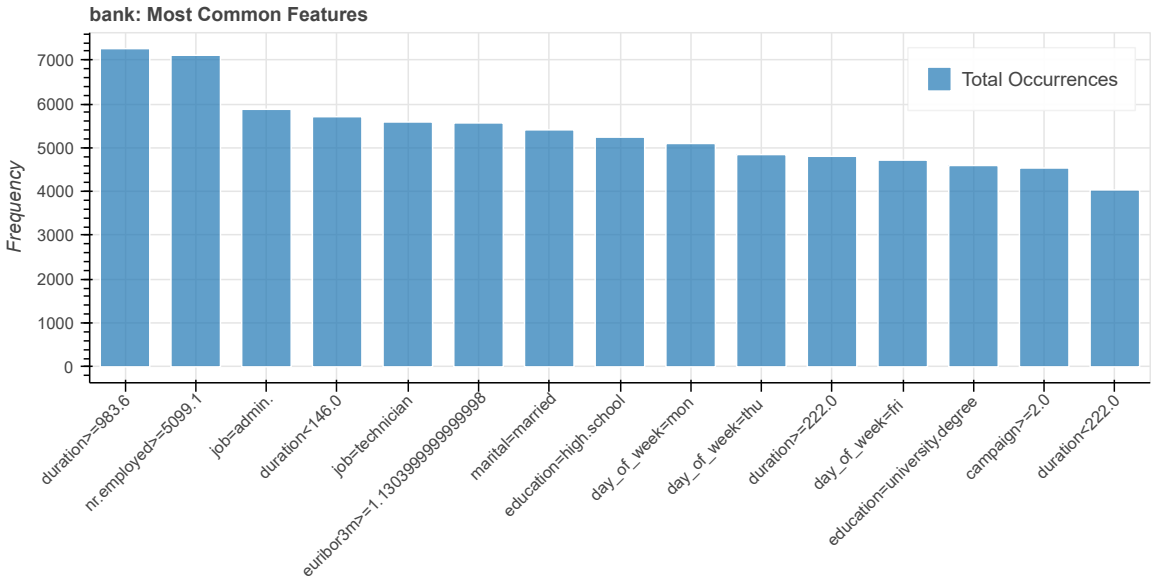
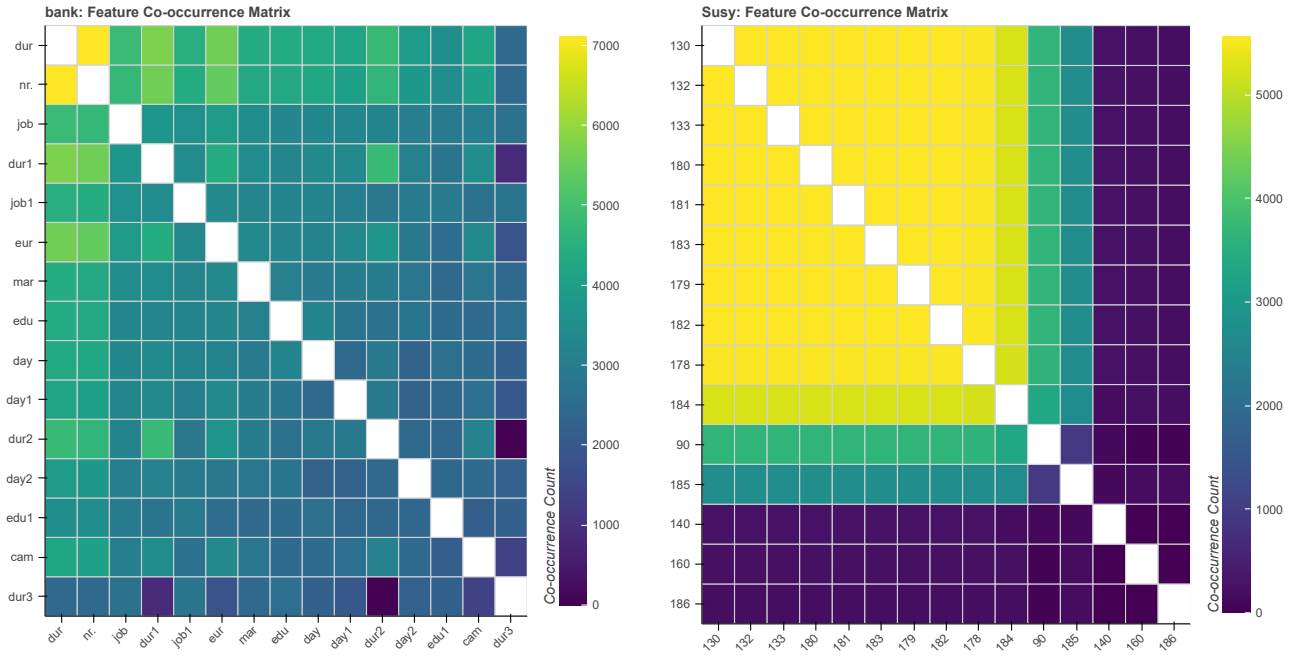


Figure 10: Feature occurrence bar chart for the Bank dataset.

Feature co-occurrence. Figure 11 displays a co-occurrence matrix for both datasets. Each cell (i, j) records how often features i and j appear together in the same rule list; the brighter the cell, the more frequent the co-occurrence.

In the `susy` figure, a compact 9×9 square stands out: the same nine variables are not only frequent but also used together. That tight grouping leaves limited space for using different features, which is consistent with the smaller number of distinct lists we observed earlier. The pattern for `bank` is looser. Two key features pair with many other secondary features, allowing CORELS to construct a greater variety of rule-list combinations than in `susy`.



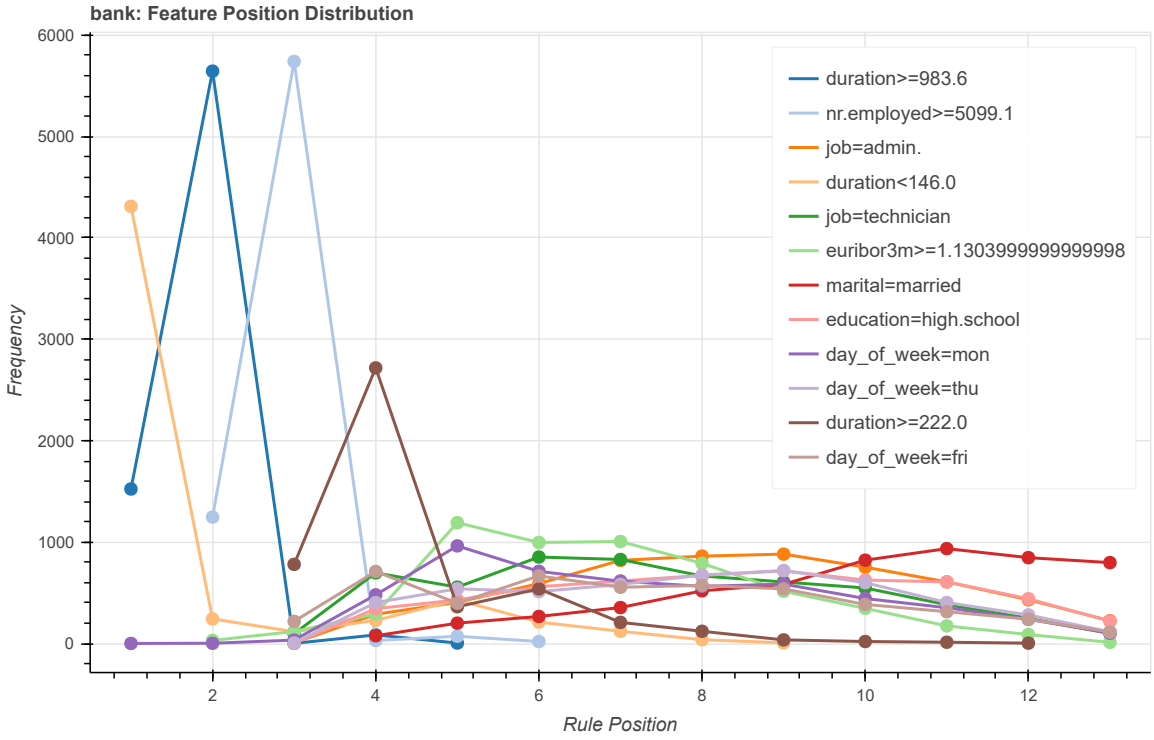
(a) Bank dataset (b) SUSY dataset

Figure 11: Feature co-occurrence matrices for the Bank and SUSY datasets, showing patterns of features appearing together in rule lists.

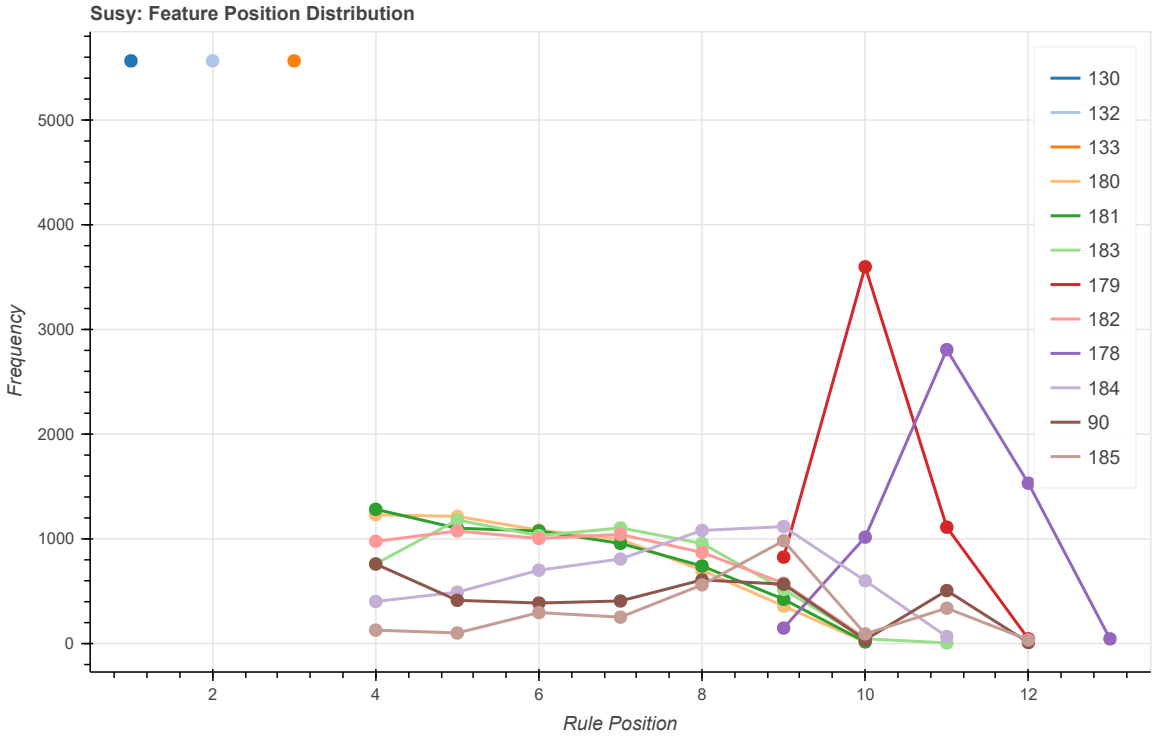
Feature positions. Figure 12 shows, for each position in the list (horizontal axis), how often a given feature occupies that slot (vertical axis counts over all unique lists). A rule list processes an instance top-down: every case is tested against rule 1, the remainder against rule 2, and so on. Consequently, bars on the left represent features that influence most predictions, whereas bars on the right correspond to decisions that affect only those instances not already handled by earlier rules.

In the `bank` panel, two variables dominate the first position and continue to trade places across positions 2-4 together with two more features, after which the histogram flattens. From position five onward, no single feature stands out. A variety of features can fill the tail without losing much accuracy. That positional flexibility matches the large number of distinct lists found earlier.

The `susy` panel is far more rigid. Exactly three features occupy the first three positions in a fixed order. The next six positions recycle the same small set of features, shuffling their order. The pattern is clear: the crucial decisions are done before position 4. Such a hierarchy explains why SUSY produces far fewer unique near-optimal lists than Bank.



(a) Bank dataset



(b) SUSY dataset

Figure 12: Feature position histograms

Taken together, the three visualisations show why the Rashomon sets of our two datasets differ so much. For `susy`, almost all predictive power sits in a tight nine-feature bundle that always appears at the very front of the list, leaving few opportunities to shuffle or substitute rules; even a generous accuracy tolerance θ therefore admits only a modest number of distinct lists. In `bank`, the key information is carried by just two indispensable features, and those pair flexibly with many secondary features in many positions, so a wide variety of rule orders remain near-optimal. Put differently, how many models fit within a given θ depends not only on the tolerance itself but also on how concentrated (as in `susy`) or dispersed (as in `bank`) the informative features are.

4.2.3 Rule-list complexity analysis

In this section, we compare the complexity of the rule lists mined in our two experimental setups. *Experimental setup 2* forces CORELS to search at every list length $k \in \{2, \ldots, 13\}$, while *setup 1* lets the solver search freely up to $k_{\max} = 13$. In both cases, Figure 13 (setup 2) and Figure 14 (setup 1) plot, for each dataset, the highest F_1 score attained at every observed length. The horizontal axis shows rule complexity (number of rules); the vertical axis shows F_1, chosen because it balances precision and recall and therefore gives the most complete single-metric picture of performance. Our aim is to see how much complexity is required before each dataset reaches its near-optimal F_1.

variable-maximum-rule-list-length (setup 2). In Mushroom and SUSY a near-optimal list emerges almost immediately: by $k = 2$ their F_1 is close to the maximum, and from $k = 4$ onward the curve is essentially flat. Adult and Bank follow the same shape, although the improvement from $k = 2$ to $k = 4$ is a little steeper, suggesting a small but tangible gain from a few extra rules. IJCNN1 is the outlier. Lists shorter than four rules yield noticeably lower F_1; only once $k \geq 4$ does performance keep stable. This indicates that four of the five problems can be solved well with two to three rules, whereas IJCNN1 clearly needs more rules. Therefore, we could say that IJCNN1 is the most complex dataset.

Constant-maximum-rule-list-length (setup 1). Because CORELS is allowed to grow lists all the way to thirteen rules, one might expect most solutions to hit that limit. Instead, the length distribution is surprisingly varied. In `bank` and `adult`, many high-scoring rule lists have just four rules or fewer. The reason for this is the small length-penalty λ used by CORELS. Every extra rule slightly lowers the score, so whenever a shorter list performs almost as well as a longer one the solver prefers the simpler option. `Mushroom` and `susy` show the opposite: virtually all of their accepted lists are nine rules or longer, yet these long variants perform nearly identically, suggesting that extra rules add little but are not penalised enough to be trimmed. For `ijcnn1`, the earlier seen pattern repeats. Only long lists (11-13 rules) reach competitive F_1, reinforcing the view that this dataset genuinely needs deeper rule lists to perform near-optimal.

Our complexity analysis reveals a clear pattern: while four of the five datasets achieve near-optimal performance with just 2-4 rules, IJCNN1 requires more complex rule structures to reach a near-optimal F_1 score.
More extensive plots are found in Appendix A.3.

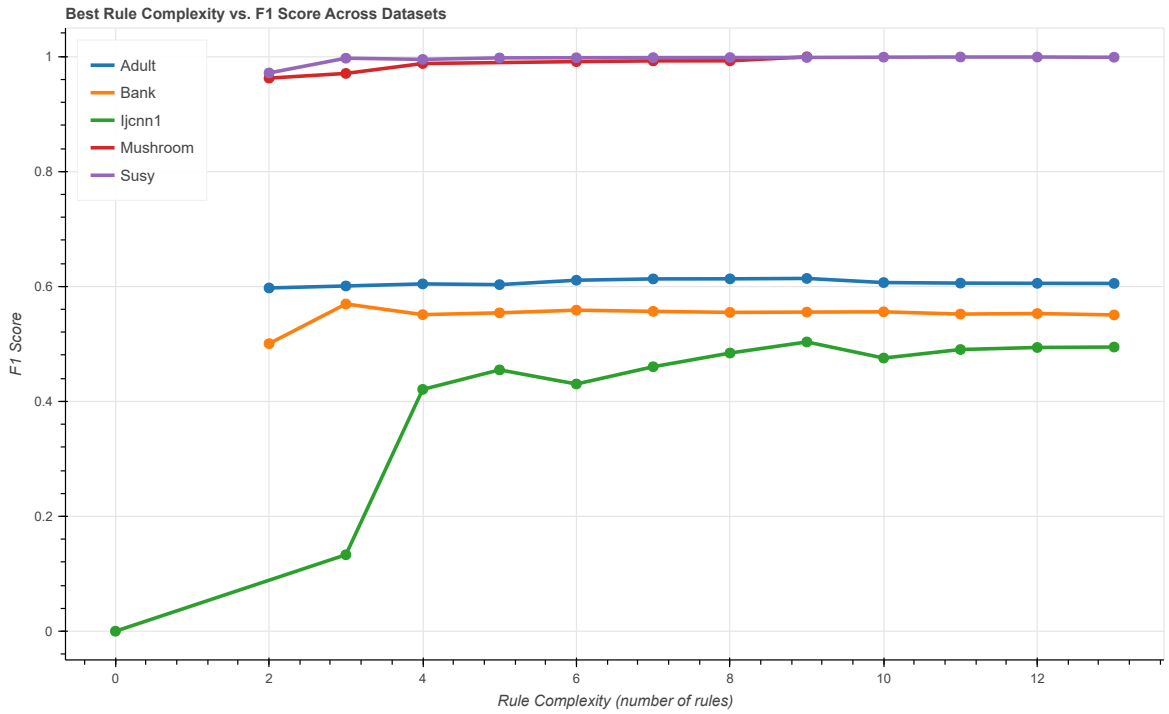


Figure 13: Best F_1 vs. rule complexity ($k = 2 \ldots 13$) — variable-length search.

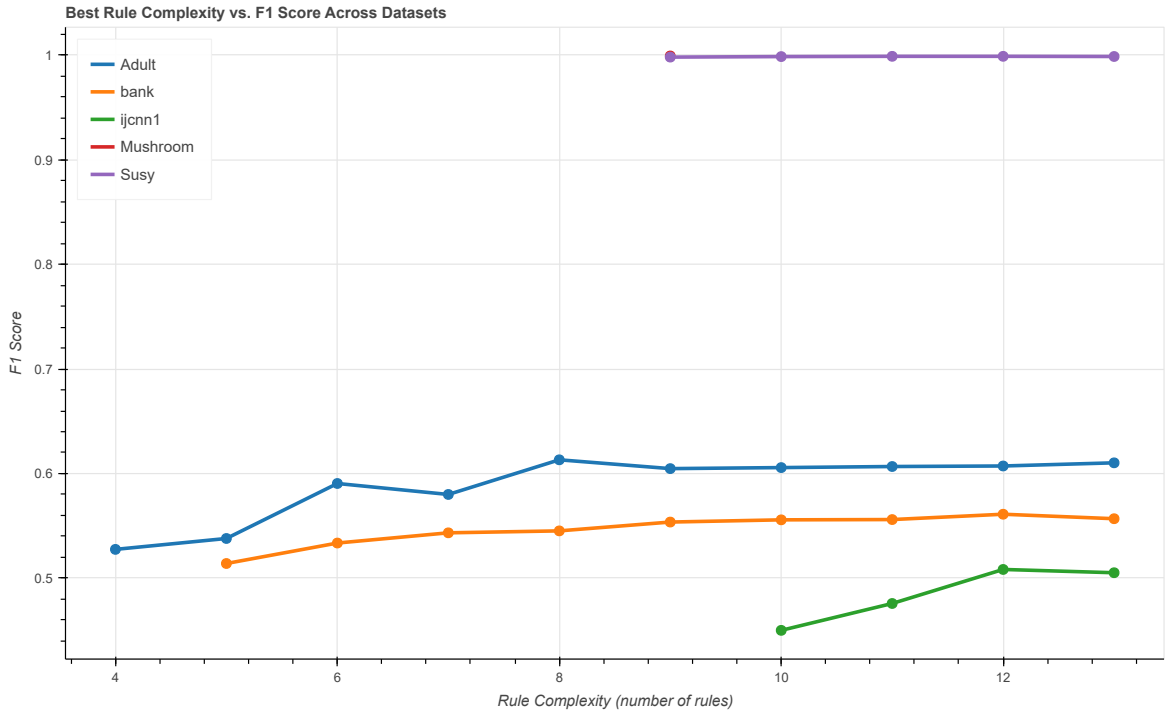


Figure 14: Best F_1 vs. rule — constant-length search.

5 Discussion

These are three points of discussion:

1. We lacked the memory to run CORELS on the full, high-dimensional benchmarks, so we trimmed both rows and features to keep the experiments tractable. This down-scaling makes the data sets comparable, but the absolute Rashomon sizes reported here could change once all features and instances are restored. Because the Random-Forest filter retained the *most* predictive features, the attributes we dropped were less informative. Adding them back would enlarge the hypothesis space and likely introduce many additional rule lists. So the total number of near-optimal models would rise, while the qualitative patterns (such as `ijcnn1` needing deeper lists) should remain the same.

2. For most variable combinations, the sample size that SamRuLe asked for was actually bigger than the original training set of the Mushroom dataset. In other words, no sampling happened at all for those combinations. Because of this issue, we had to exclude the Mushroom dataset from some analyses.

3. CORELS uses a loss function that is based on accuracy. Especially for imbalanced datasets this could be a problem. With a low k value it sometimes returns the easy "always predict 0" rule, which nails accuracy but has no recall. A better fit for many real-world problems would be F1-score. It would be interesting to see if it is possible to switch the loss function to F1 (or something similar)without blowing up running time.

6 Conclusion

This thesis examined whether the variability produced by SamRuLe can be used to understand a dataset's Rashomon set. Rule lists combine interpretability with the ability to be mined in large quantities, making them an ideal lens for this study. Our experiments deliver two main contributions: a cross-dataset comparison of Rashomon-set growth and a three-part rule list structure analysis that looks at model complexity, feature usage, and the number of distinct lists uncovered. The Rashomon set growth curves proved stable across folds and revealed clear dataset differences. Balanced datasets (`mushroom`, `susy`) reached a full Rashomon set at tiny tolerances, while imbalanced data (`adult`, `bank`, `ijcnn1`) expanded much more slowly. Rule-list diversity varies by dataset: when the predictive signal is spread over many interchangeable features, CORELS uncovers a rich mix of near-optimal lists, whereas a tight feature core yields less variety. Most datasets reach their best F_1 with very short lists, but `ijcnn1` stands out. It requires more complex rule lists to achieve near-optimal performance, indicating being a more complex dataset. Overall, the difference in near-optimal rule lists found is helpful in characterising the Rashomon set and providing insights about their dataset.

References

[ALSA+18] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 18(234):1–78, 2018.

[Bre01] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16, 08 2001.

[Coh95] William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML–95)*, pages 115–123, San Francisco, CA, USA, 1995. Morgan Kaufmann. Introduces the RIPPER algorithm.

[PV24] Leonardo Pellegrina and Fabio Vandin. Scalable rule lists learning with sampling. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, page 2352–2363. ACM, August 2024.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Riv87] Ronald L Rivest. Learning decision lists. *Machine learning*, 2:229–246, 1987.

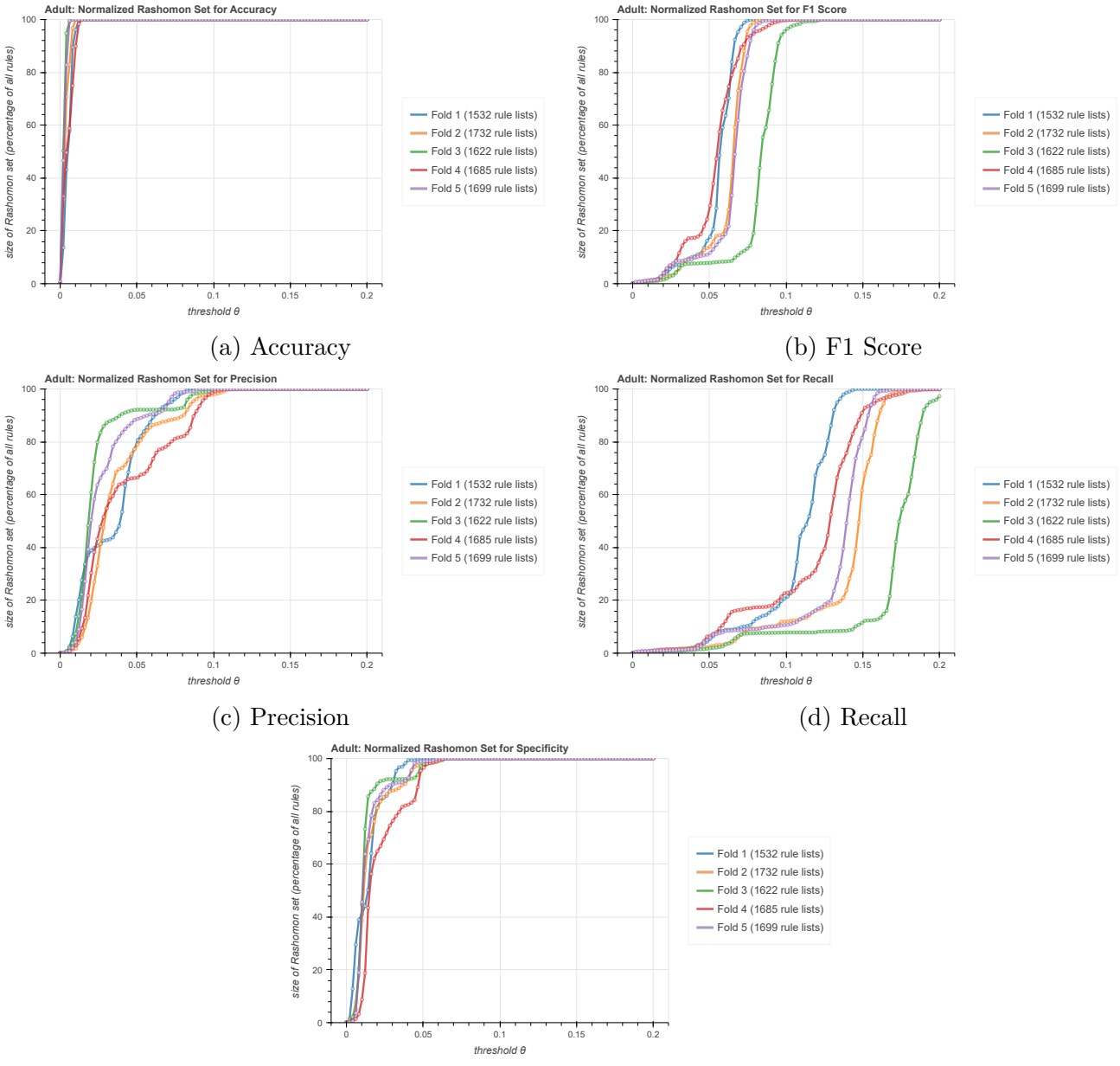
[Rud19] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, 2019.

[SL09] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45:427–437, 2009.

[SR19] Lesia Semenova and Cynthia Rudin. A study in rashomon curves and volumes: A new perspective on generalization and model simplicity in machine learning. *CoRR*, abs/1908.01755, 2019.

A Additional Plots

A.1 Plots for Section 4.1.1



(a) Accuracy

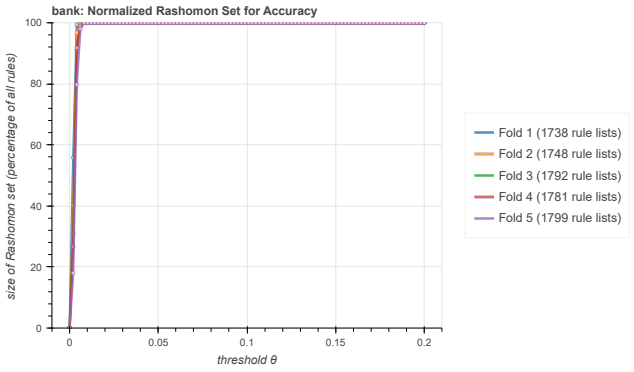
(b) F1 Score

(c) Precision

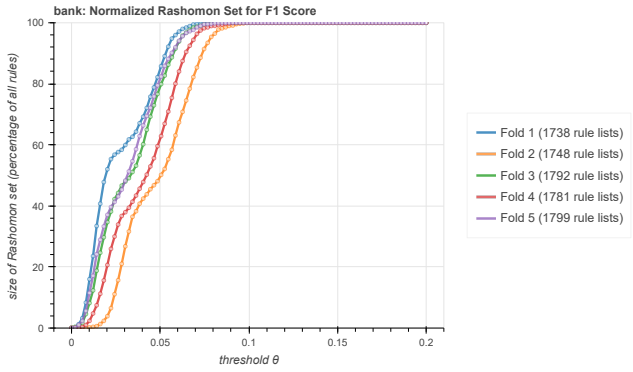
(d) Recall

(e) Specificity

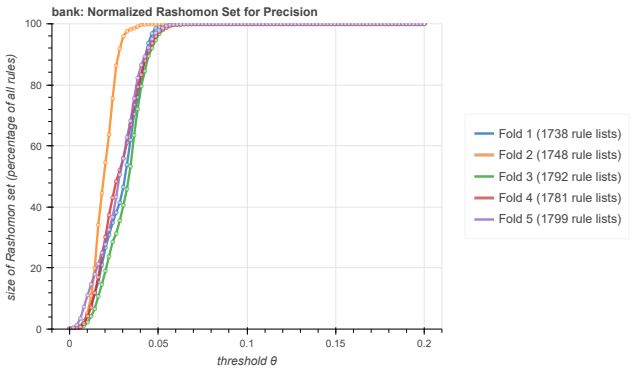
Figure 15: Normalized metrics for Adult dataset (constant k)



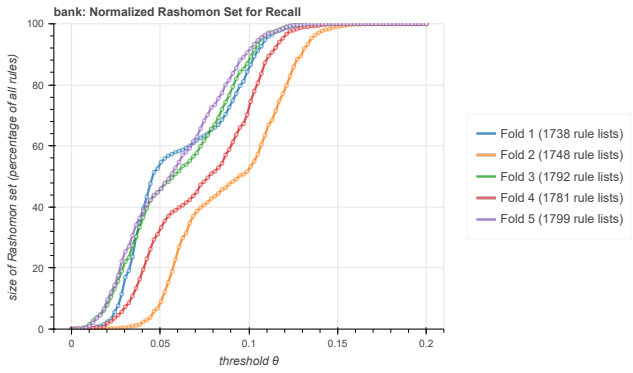
(a) Accuracy



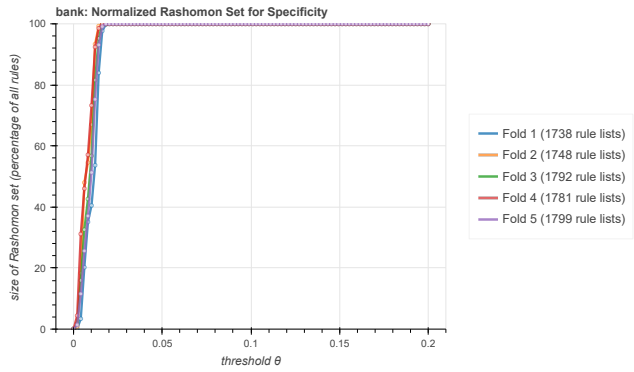
(b) F1 Score



(c) Precision

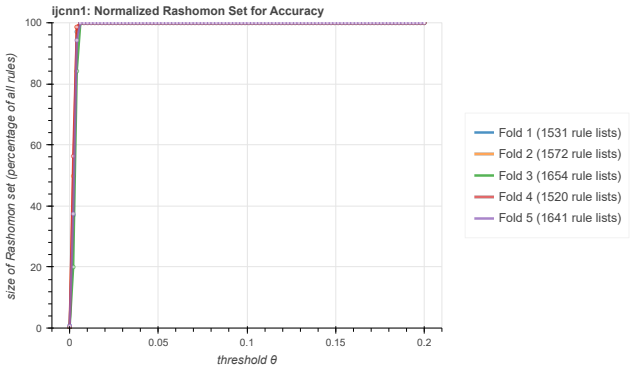


(d) Recall

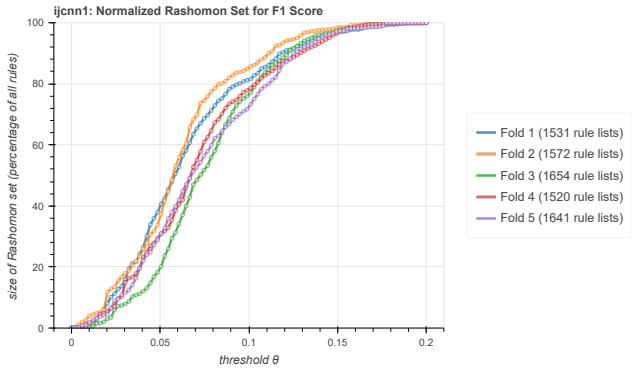


(e) Specificity

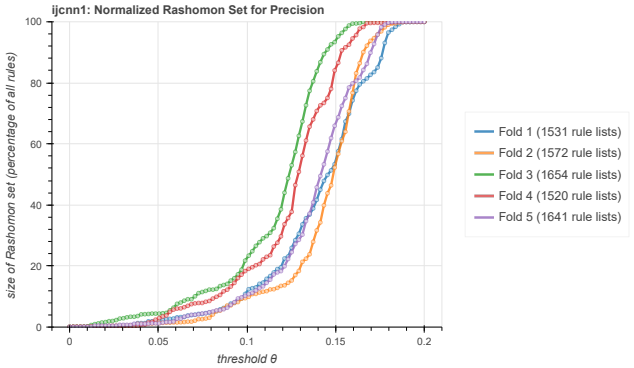
Figure 16: Normalized metrics for Bank dataset (constant k)



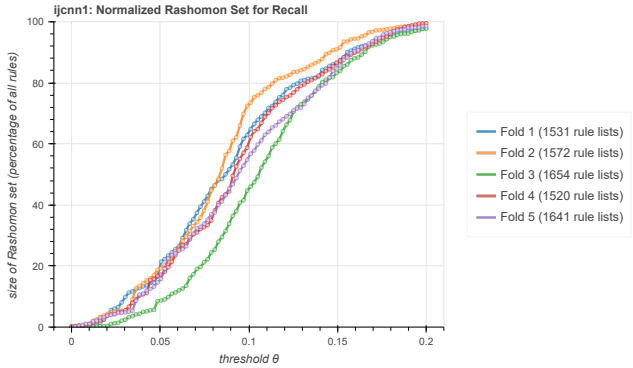
(a) Accuracy



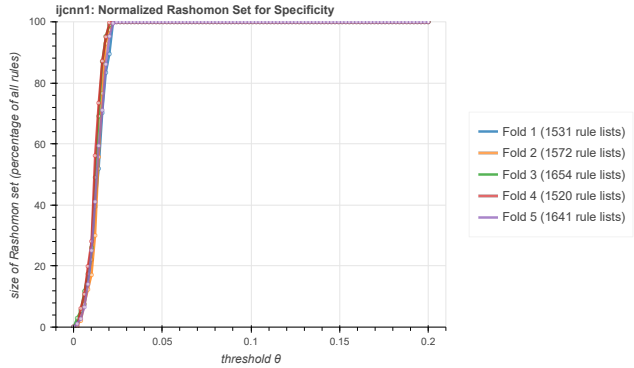
(b) F1 Score



(c) Precision

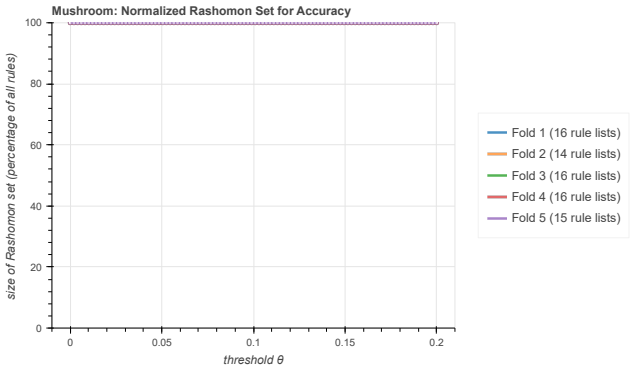


(d) Recall

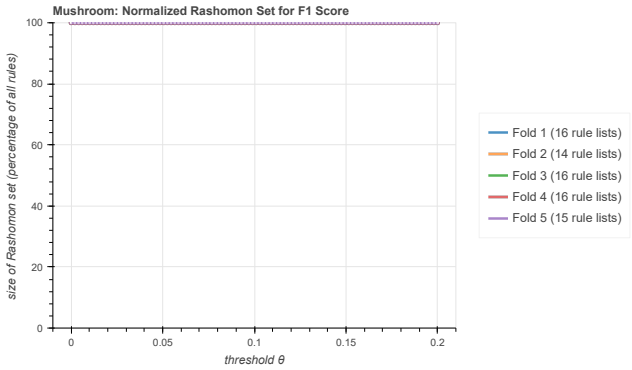


(e) Specificity

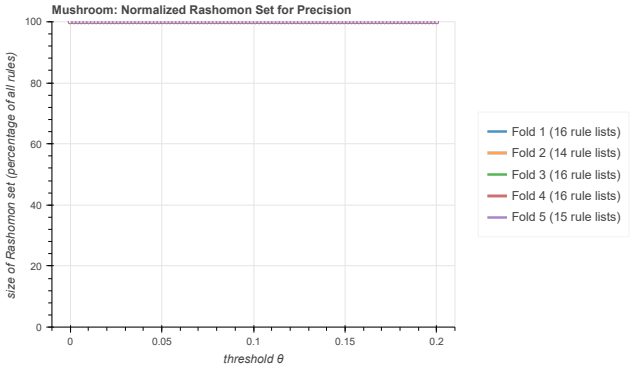
Figure 17: Normalized metrics for IJCNN1 dataset (constant k)



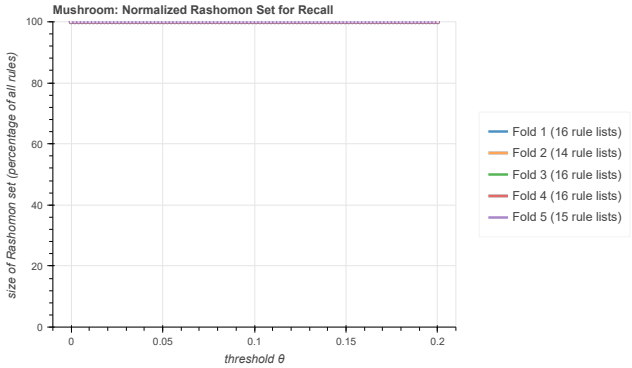
(a) Accuracy



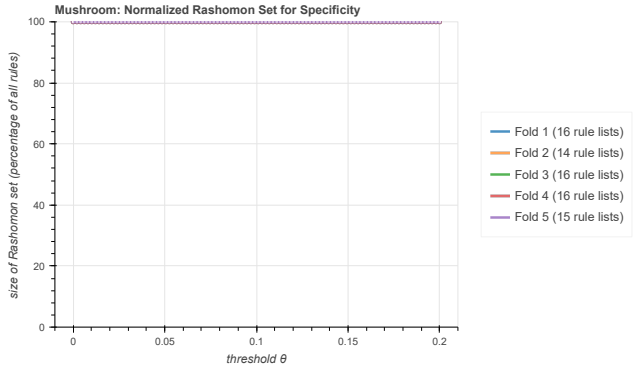
(b) F1 Score



(c) Precision

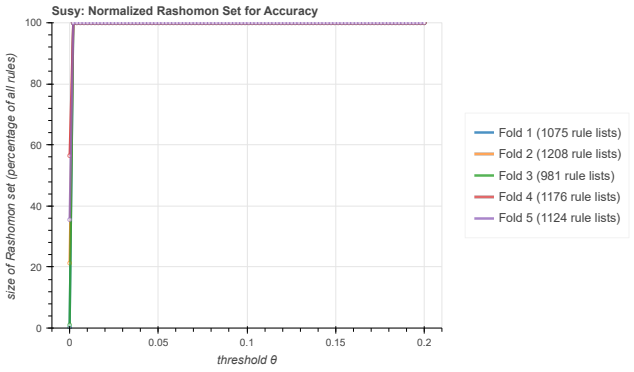


(d) Recall

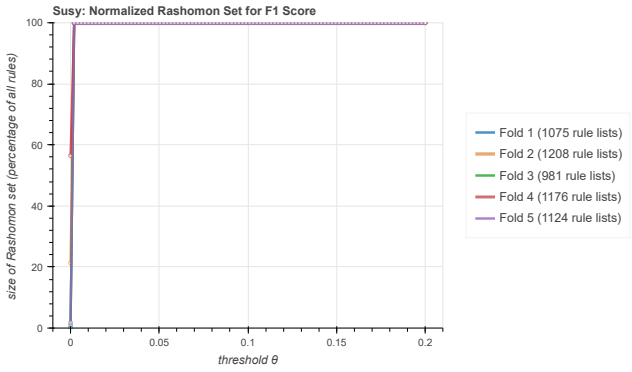


(e) Specificity

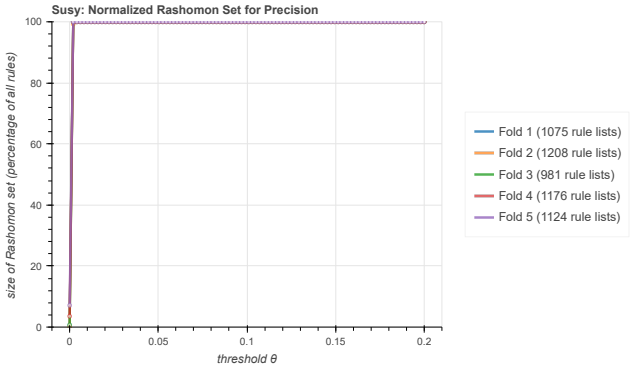
Figure 18: Normalized metrics for Mushroom dataset (constant k)



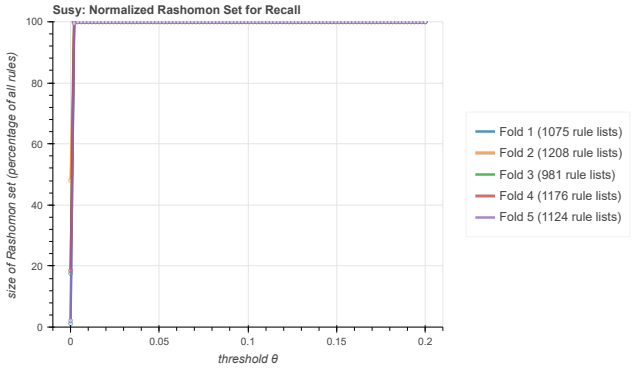
(a) Accuracy



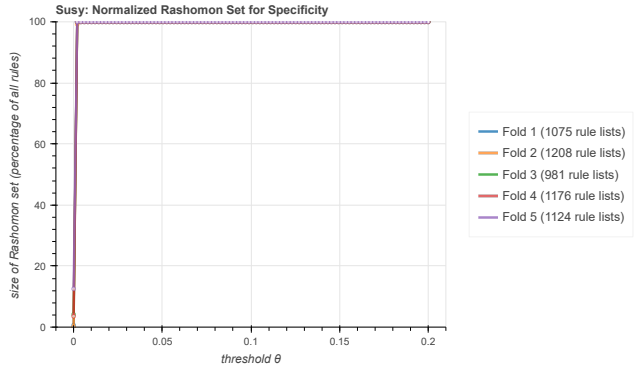
(b) F1 Score



(c) Precision



(d) Recall

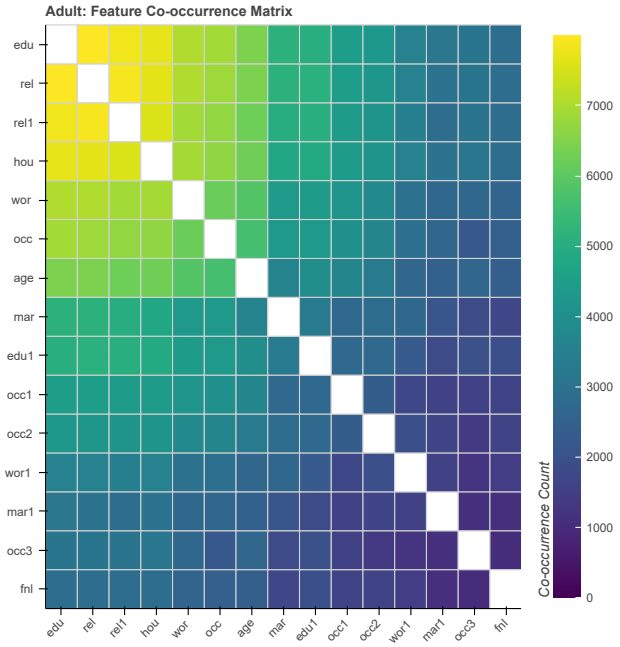


(e) Specificity

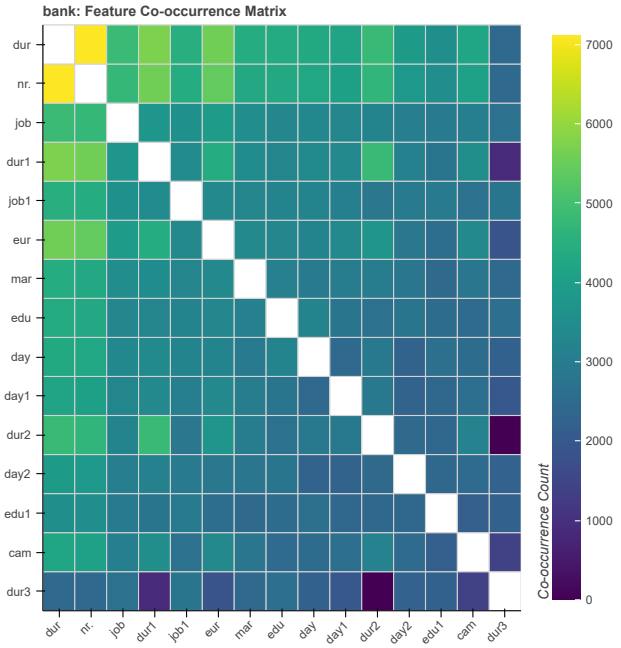
Figure 19: Normalized metrics for SUSY dataset (constant k)

A.2 Plots for Section 4.2.2

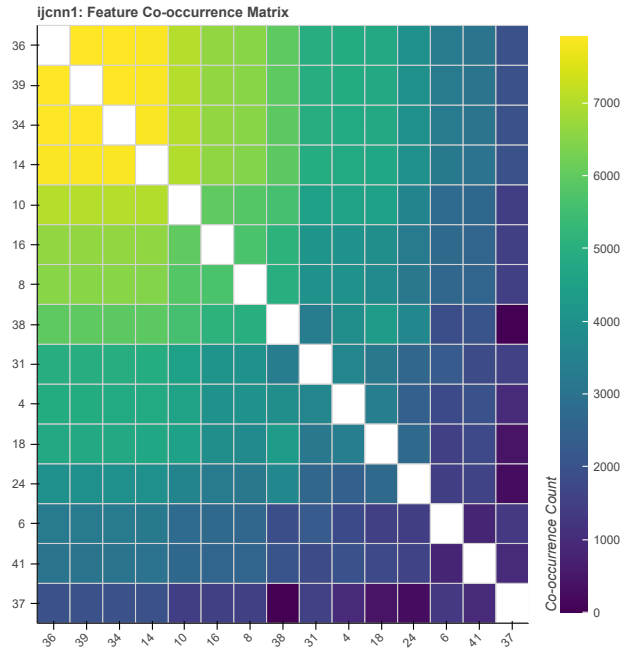
A.2.1 Feature Co-occurrence Heatmaps



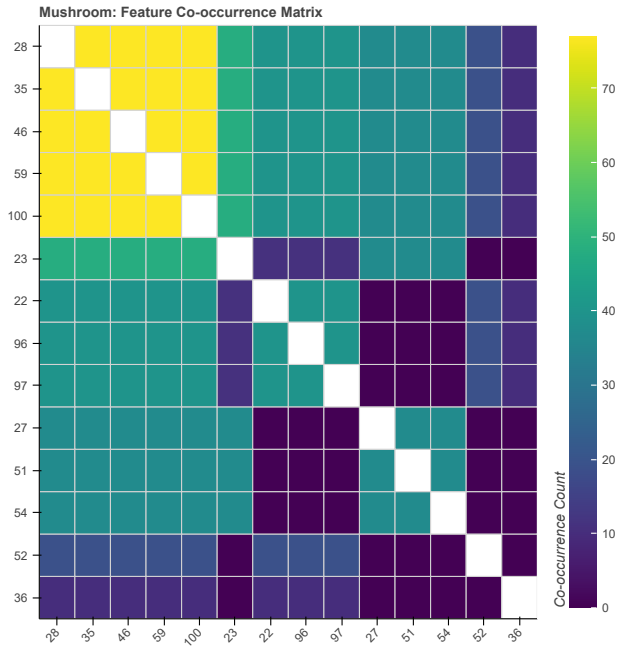
(a) Adult dataset



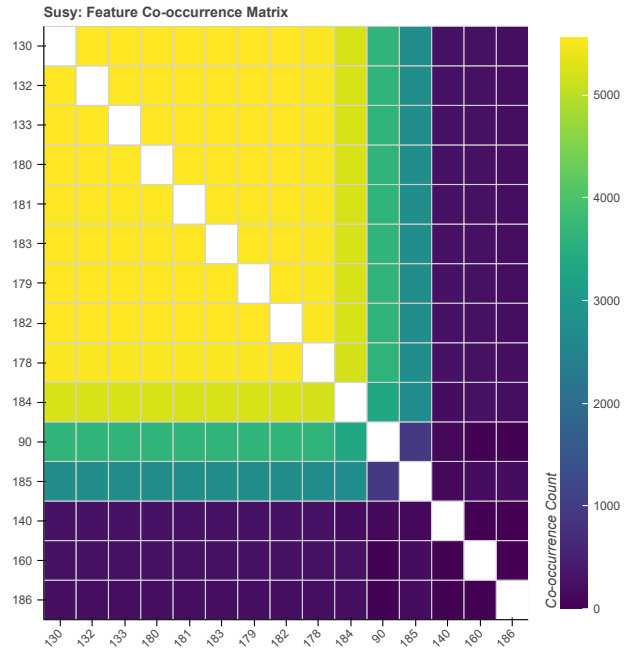
(b) Bank dataset



(a) IJCNN1 dataset



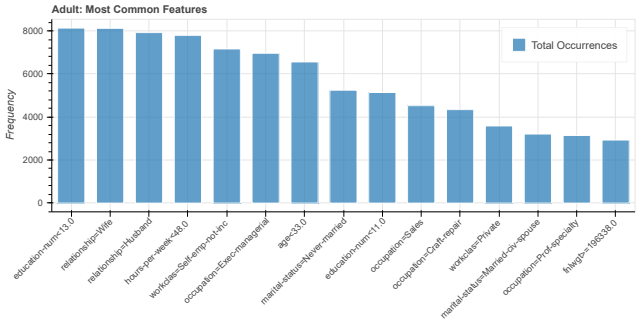
(b) Mushroom dataset



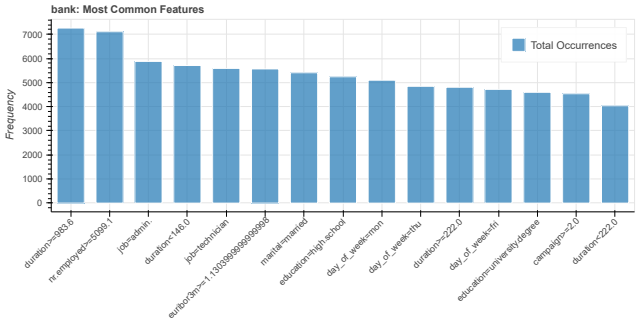
(c) SUSY dataset

Figure 21: Feature co-occurrence heatmaps for all datasets

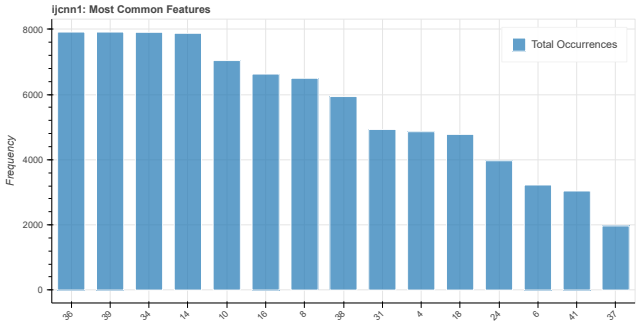
A.2.2 Feature Frequency Plots



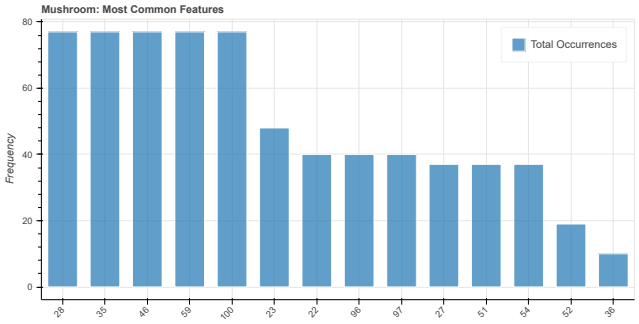
(a) Adult dataset



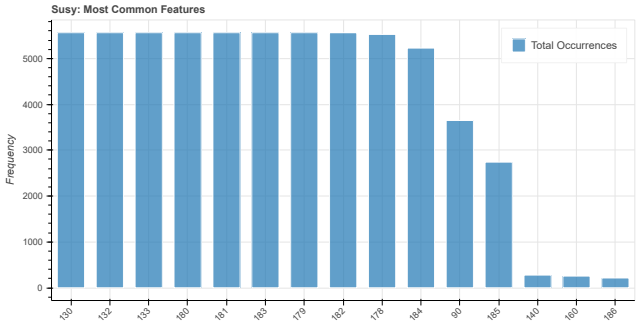
(b) Bank dataset



(c) IJCNN1 dataset



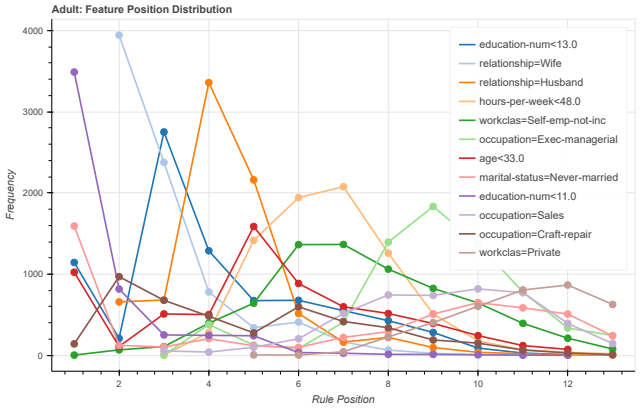
(d) Mushroom dataset



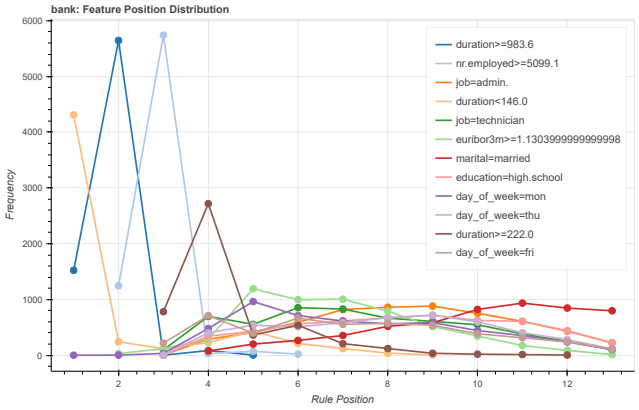
(e) SUSY dataset

Figure 22: Feature frequency plots for all datasets

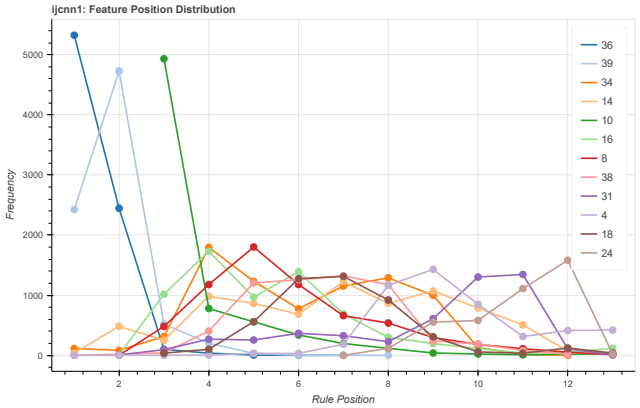
A.2.3 Feature Position Plots



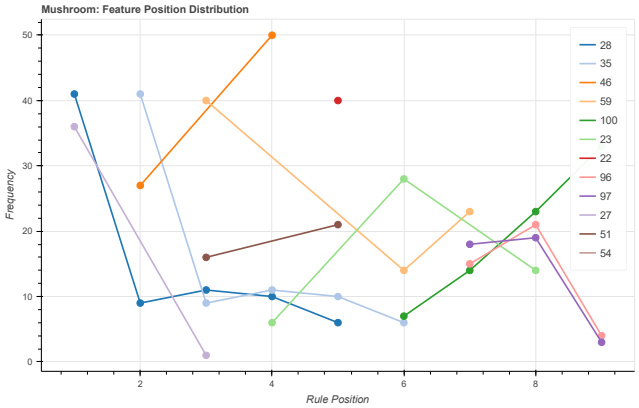
(a) Adult dataset



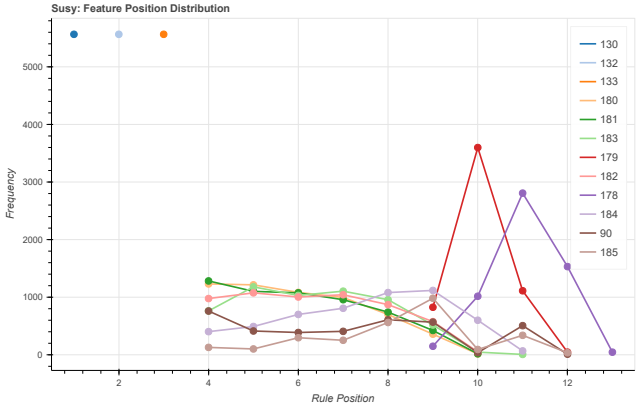
(b) Bank dataset



(c) IJCNN1 dataset



(d) Mushroom dataset



(e) SUSY dataset

Figure 23: Feature position plots for all datasets

A.3 Plots for Section 4.3.2

A.3.1 Dataset Comparison Complexity Plots

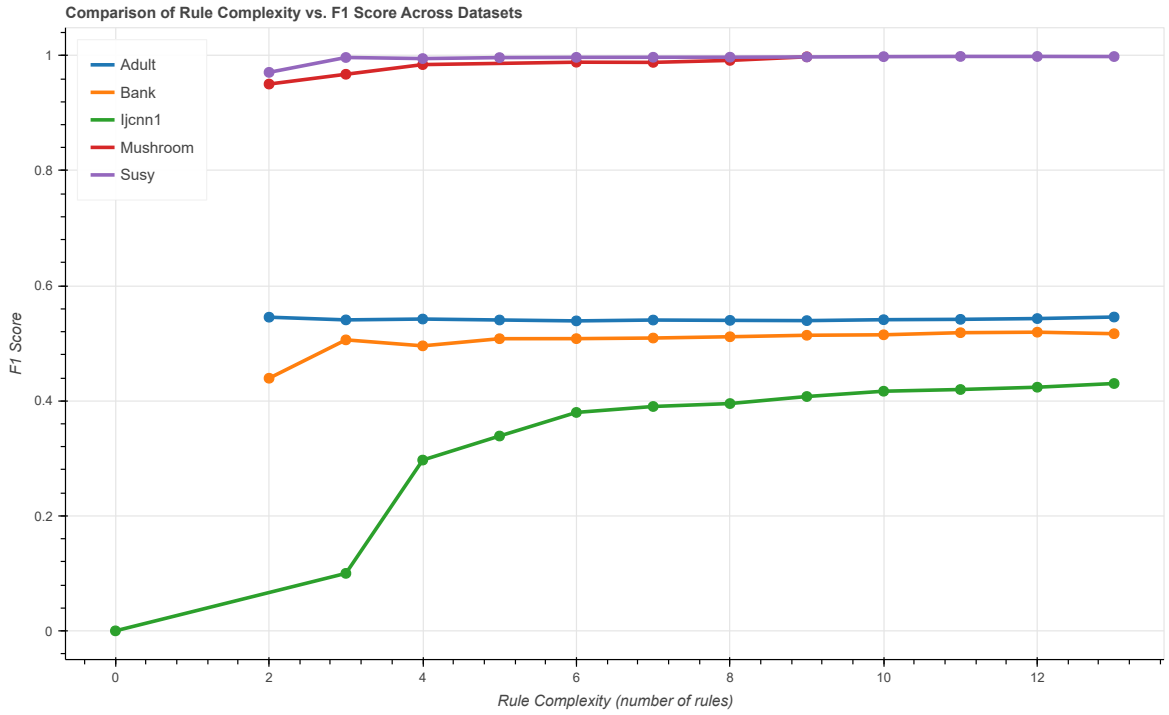


Figure 24: Experimental setup 2: average F1 score complexity comparison across all datasets

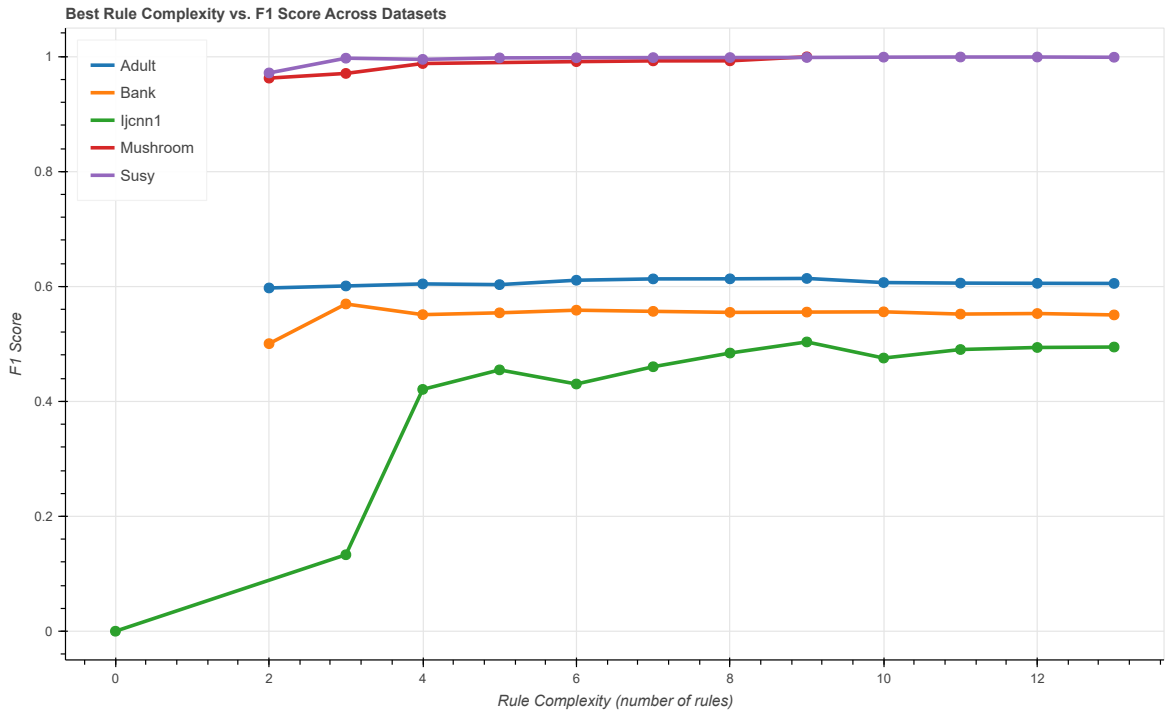


Figure 25: Experimental setup 2: Best F1 score complexity comparison across all datasets

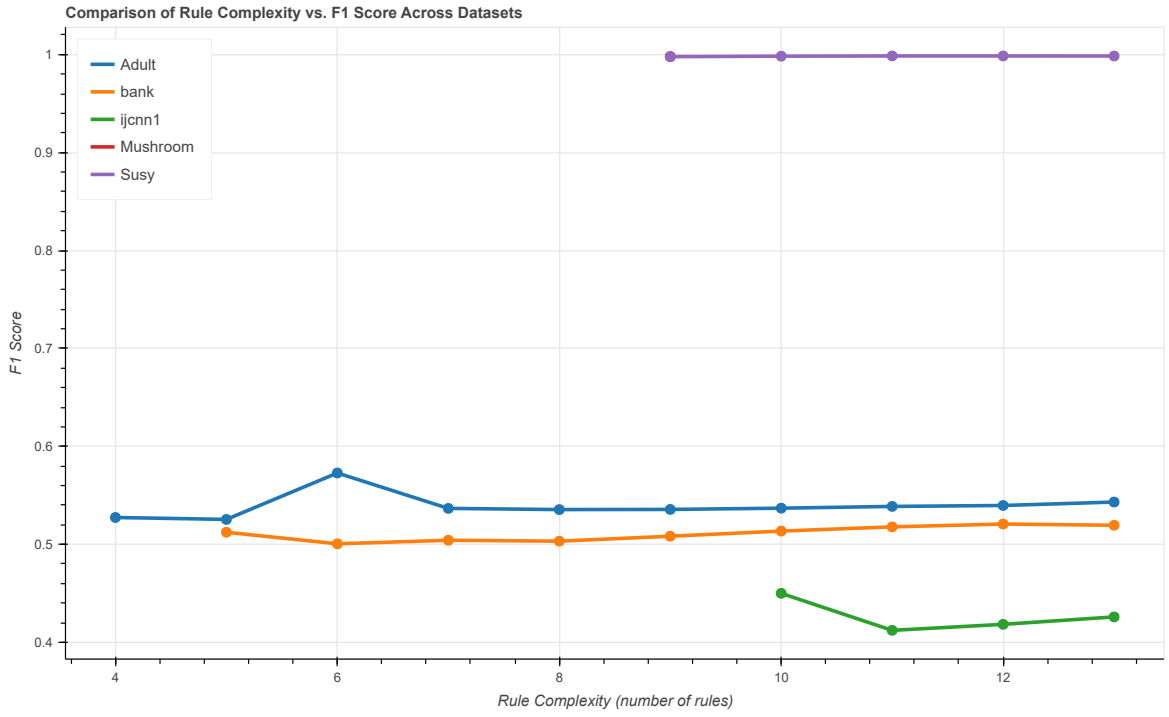


Figure 26: Experimental setup 1: average F1 score complexity comparison across all datasets

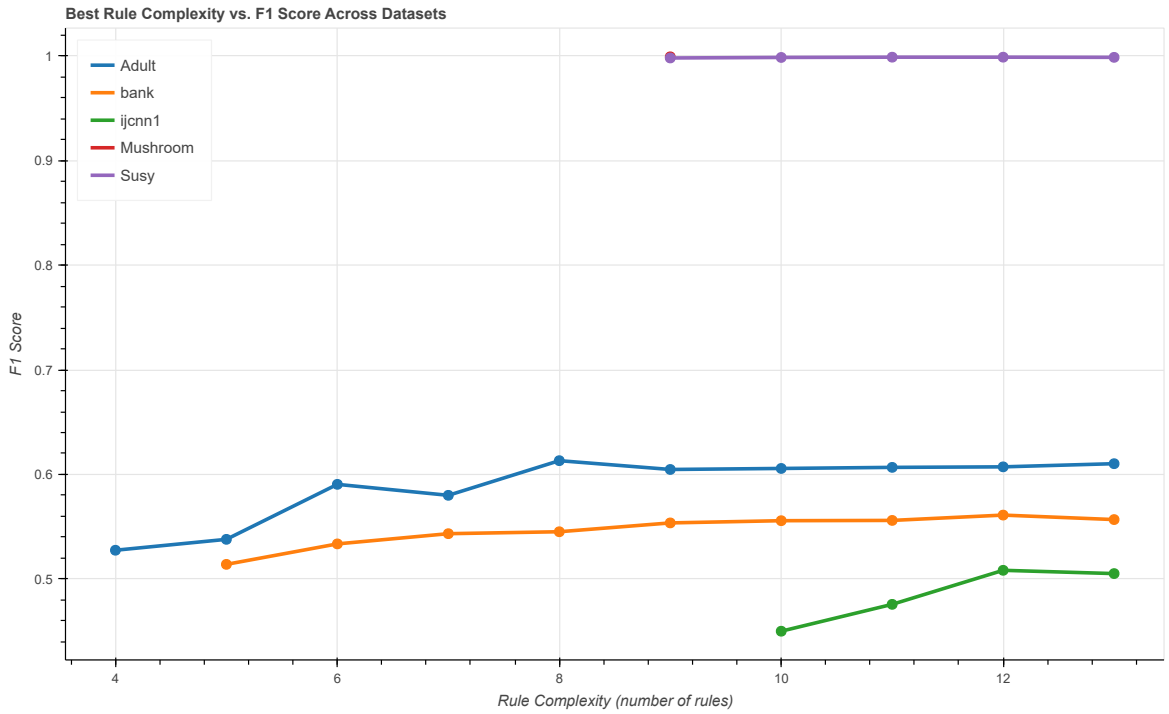
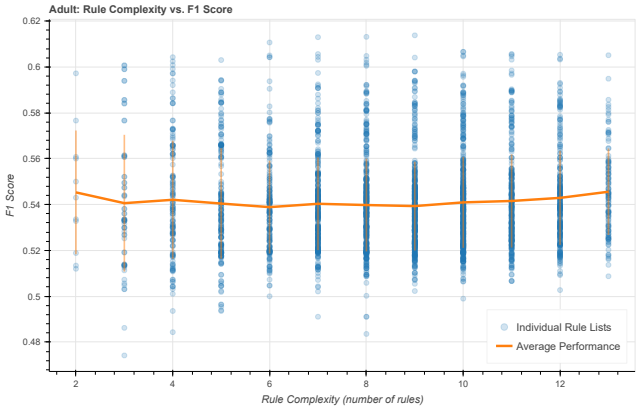
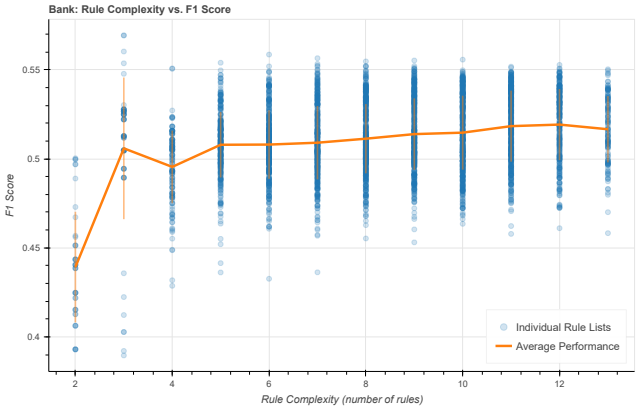


Figure 27: Experimental setup 1: Best F1 score complexity comparison across all datasets

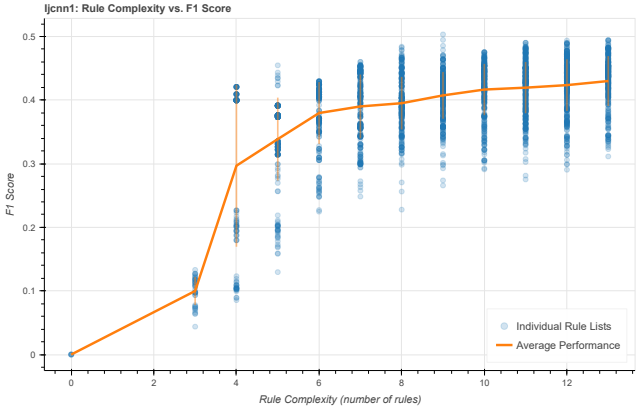
A.3.2 Individual Dataset Complexity Plots



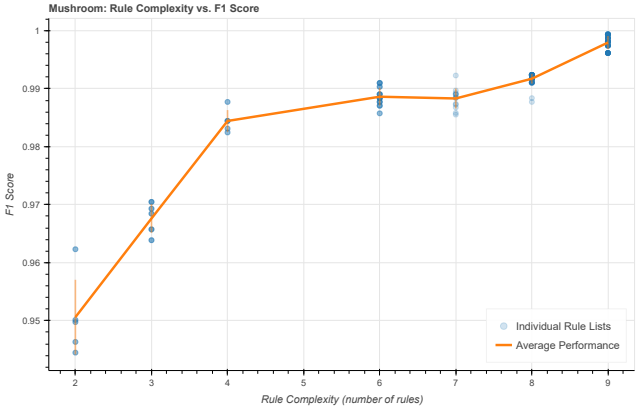
(a) Adult



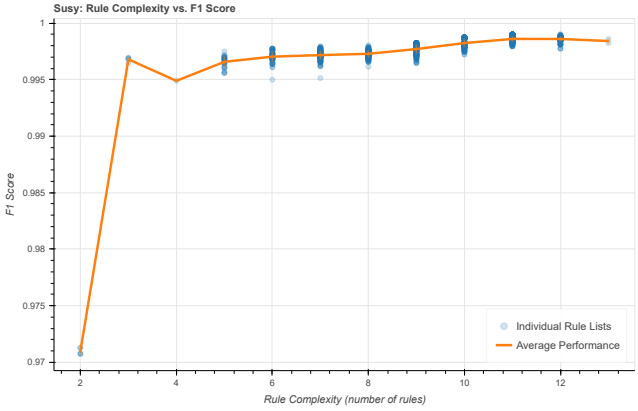
(b) Bank



(c) IJCNN1



(d) Mushroom



(e) SUSY

Figure 28: F1 score complexity plots for individual datasets ($k = 2 \dots 13$) — experimental setup 2