Supplementary material for AISTATS 2020
LIBRE: Learning Interpretable Boolean Rule Ensembles

A THE BASE METHOD STEP BY STEP

In this section, we show in detail the main steps of the base algorithm, by using a concrete example.

Consider the scenario of forecasting the failure condition of an IT system from two values representing the CPU and main memory (MEM) utilization, as depicted in the first two columns of table 1. We assume that CPU and MEM are continuous features with values in the domain $[0, 100]$. The state of the system is described by a binary *Label*, where 1 represents a system failure. The example reports eight records, of which two are failures.

	CPU	MEM	r_1	r_2	String	Label
t_1	95	10	3	1	110 01	1
t_2	80	10	1	1	011 01	0
t_3	81	85	2	2	101 10	1
t_4	10	85	1	2	011 10	0
t_5	10	10	1	1	011 01	0
t_6	82	10	2	1	101 01	0
t_7	85	10	2	1	101 01	0
t_8	81	10	2	1	101 01	0

Table 1: Original values from CPU and MEM, their mappings to discrete ranges (r_1, r_2), binary encoding, and binary label.

A.1 Discretization And Binarization

The first operation to do is discretization. Assume the discretization algorithm identifies three intervals for CPU and two intervals for MEM, as follows. CPU: $[0, 81), [81, 95), [95, max)$. MEM: $[0, 85), [85, max)$. We can now map the original values to integer values over the ranges (1, 2, 3) and (1, 2), as shown in columns r_1, r_2, respectively. The resulting discretized records are then mapped to (inverse one-hot encoded) binary strings of five bits, as recorded in the *String* column. We also define a partial order relation between binary records, such that $\mathbf{x} \leq \mathbf{x}' \iff \mathbf{x} \bigwedge \mathbf{x}' = \mathbf{x}'$. Moreover, the application of inverse one-hot encoding ensures that the relation between input features and labels is monotone, according to definition 2.2 in the main paper. We can give you an intuition through a simple example: consider two binary strings 011 and 110; we see that $011 \not\leq 110$ and $110 \not\leq 011$, so the relation always holds, independently from the label.

A.2 Learning The Boundary

Consider the first positive sample t_1 with string 110 01. An exhaustive search strategy would explore all possible flipping alternatives for the most general conflict-free binary strings. If, for example, we flip-off the first bit we obtain 010 01 $<= t_2$: we have therefore a conflict. If, for example, we keep the first bit at 1 and flip-off the second bit, we obtain 100 01, which is in conflict with $t_6 - t_8$. Finally, if we flip-off the last bit, we obtain 110 00, which has no conflict: this is a candidate boundary point. If we repeat the same procedure for t_3, after flipping-off the third bit, we obtain another boundary point 100 10.

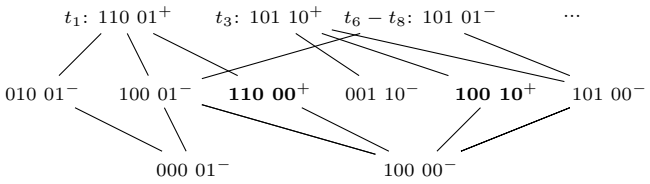


Figure 1: Partially ordered set created from the records in table 1.

Figure 1 shows the partially ordered set corresponding to table 1. At the beginning, the nodes at the top are the ones for which we know the label represented with a superscript symbol + and − for positive and negative, respectively. They can be seen as maximally-specific rules. If we take as target the positive class, we move inside the Boolean lattice by flipping-off positive bits, starting from the positive binary samples, and go down to find binary elements – located on the boundary – that divide positive and negative samples. While we navigate the Boolean lattice, nodes are labelled according to the cover test against the negative samples. As soon as a conflict is found, we can avoid going down from that node, but there is still the possibility to explore that path from another binary sample. This recursive procedure corresponds to up-and-down movements in the lattice. However, if at each iteration we are able to select the best candidate bit and to avoid conflicts, we only allow steps down in the Boolean lattice. We use the heuristic described in the main paper to choose the best candidate bit to flip-off.

A.3 A Practical Example

Consider again the example in table 1. Since at the beginning $\mathcal{S} = \mathcal{T}$, we will only report $|\mathcal{T}_i^0|$. For the first positive record t_1 =110 01, we have: $\mathcal{F}_1^0 = \{01101, 01110\}$, $\mathcal{F}_2^0 = \{10101\}$, $\mathcal{F}_5^0 = \{01110\}$. We have therefore: $d_l(t_1, \mathcal{F}_1^0) = 1$, $d_l(t_1, \mathcal{F}_2^0) = 1$, $d_l(t_1, \mathcal{F}_5^0) = 2$. We already know that flipping-off either the first or the second bit to 0 would lead to a conflict: thus, we directly flip-off the fifth bit to obtain the boundary point 110 00, independently from the value of $|\mathcal{T}_5^0|$. Element 110 00 is added in the set of boundary points $\mathcal{A}$.

For the second positive record t_3 = 101 10, we have: $\mathcal{F}_1^0 = \{01101, 01110\}$, $\mathcal{F}_3^0 = \emptyset$, $\mathcal{F}_5^0 = \{01110\}$. We have therefore: $d_l(t_1, \mathcal{F}_1^0) = 1$, $d_l(t_1, \mathcal{F}_3^0) = undefined$, and $d_l(t_1, \mathcal{F}_4^0) = 1$. Although $i = 3$ induces a distance from an empty set, since we know that flipping-off other indexes generates conflicts, we can immediately label 100 10 as boundary point and add it to $\mathcal{A}$.

A.4 From Boundary Set To Rules

At the end of the previous phase, we obtain the boundary set $\mathcal{A} = \{11000, 10010\}$. In this case, each boundary point covers only one distinct positive sample, therefore the union of the two points covers all the set of positive samples and both points are kept after the regularization. Let's suppose to follow a positive set cover strategy, without early stopping condition. Then, the boundary set can be immediately mapped to the rule set shown in fig. 2.

IF CPU $\in$ [95, max)
OR CPU $\in$ [81, max) and MEM $\in$ [85, max)
THEN Label = 1
ELSE Label = 0

Figure 2: Ruleset extracted from the boundary

A.5 Differences with Muselli and Quarati (2005)

Our base method is similar to Muselli and Quarati (2005) that we took as a reference since it presents interesting properties: it is a bottom-up method, easily parallelizable, incorporating interpretability goals in the induction process. To better analyze the differences with our proposal, we refer to a technical report (Muselli and Quarati, 2004) where the authors provide more details about their *shadow clustering* algorithm. Looking at the pseudo-code (Figure 4 of Muselli and Quarati (2004)) we can firstly notice that our heuristics H_1 and H_2 are 3-length tuples (instead of 4-length) where we avoid to compute what Muselli and Quarati (2004) indicate as $|B_i^1|$ (we verified experimentally that it has no impact in practice). The main improvement is however related to the computation of the distance $d_l(p(I \cup J), D_i^0)$ which complexity is $O(nd)$. In Muselli and Quarati (2004), such distance is computed within a while loop (of length d in the worst case), for every candidate bit $i \in I$ to flip. We instead compute the distance only once outside the while loop, and update it iteratively for i_{best} only (the complexity of the update is $O(n)$ instead of $O(nd)$). Indeed, we flip one bit at a time and there is no need to re-compute the distance from scratch for the other bits. Thus, we lower the total complexity from $O(n^2d^3)$ to $O(n^2d^2)$.

B PARALLEL AND DISTRIBUTED IMPLEMENTATION

LIBRE is amenable to parallel and distributed implementations. Indeed, it processes one positive sample at a time. An exhaustive version of the `FindBoundaryPoint()` procedure is embarrassingly parallel and it is easily parallelizable on multi-core architectures: it is sufficient to spawn a UNIX process per positive sample, and exploit all available cores.

Instead, the approximate procedure, requires a slightly more involved approach. Indeed, the approximate `FindBoundaryPoint(.)` procedure processes positive records that have not yet been covered by any boundary point. Hence, a global view on the set $\mathcal{S}$ is required. We experimented with two alternatives. The first is to place $\mathcal{S}$ in a shared, in-RAM datastore, because UNIX processes – unlike threads – do not have shared memory access. The second alternative is to simply let each individual process to hold their own version of $\mathcal{S}$, thus sacrificing a global view. Our experiments indicate that the loss in performance due to a local view only is negligible, and largely out-weighted by the gain in performance, since the execution time decreases linearly with the number of spawned UNIX processes. Moreover, both $\mathcal{D}_+$ and $\mathcal{D}_-$ remain consistent throughout the whole induction phase.

LIBRE can be easily distributed such that it can run on a cluster of machines, using for example a distributed computing framework such as Apache Spark spa. This approach, called *data parallelism*, splits input data across machines, and let each machine execute, independently, a weak learner. The data splitting operation shuffles random subsets of the input features to each worker machine. Once each worker finishes to generate the local rule sets, they are merged in the "driver" machine, which eventually applies the filtering and then executes the rule selection procedure to produce the final boundary.

C LINK TO THE CODE

https://github.com/grazianomita/LIBRE

D THE IMPACT OF LIBRE'S PARAMETERS

In this section we investigate how acting on LIBRE's parameters allows to obtain specific performance-interpretability tradeoffs. We will not cover all possible parameters: in particular, we focus on the discretization threshold, $\#estimators$, and $\#features$ per estimator. The effects of α and early-stopping in weighted set cover are not reported here since their effects are well known from previous studies.

When we vary one parameter, all the others are kept fix to isolate its impact. We will also give some rules of thumb to choose them.

D.1 The Effects Of Varying The Discretization Threshold

The choice of the discretization threshold depends on the specific dataset: a threshold equal to zero means no discretization, whereas increasing the threshold is equivalent to increase the tolerance to combine consecutive ranges of values with different label distributions. In general, a zero threshold gives bad performances and results in a bigger lattice with a consequent slower training time; also a too aggressive (high) threshold is not recommended because it would lead to a huge loss of information. The most significant effects occur as soon as we start increasing the threshold: in general, F1-score improves (and eventually oscillates) up to a value after which it can eventually decrease.

The threshold affects also the number of rules and their size. In general, when there is no discretization, two extreme cases are possible: i) We might have as many rules as the number of positive examples (if their binary representation does not generate conflicts with the elements in $\mathcal{F}$) with $\#atoms = \#features$. It means that the model simply overfitted the training data. ii) We might end up with few rules with very high number of atoms (or no rules at all): the model tried to generalize positive records but it was not able to learn something meaningful because too many conflicts were present in the dataset.

From our experiments, the second option is more common (few complex rules). Again, as soon as we start increasing the threshold, the model starts to learn: the number of discovered rules increases and the number of atoms decreases, since the model is able to filter out useless features. After that, changes tend to stabilize: in our experiments, this happens when the discretization threshold is roughly between 3 and 6.

D.2 The Effects of Varying $\#estimators$ And $\#features$

We analyze how $\#estimators$ and $\#features$ affect the predictive performance and interpretability of LIBRE, by keeping fixed the remaining parameters. Results are reported for the HEART UCI dataset, but the considerations we do are quite general.

Parameter Settings. We fix a discretization threshold = 6. The search procedure optimizes the $H1$ heuristic, without applying any filtering before running weighted set cover, for which we set $\alpha = 0.7$, without applying any early-stopping condition. We vary $\#estimators \in \{1, 5, 10, 15, 20\}$ and $\#features \in \{1, 2, 3, 4, 5, 6, 7, 8\}$. We perform up to 50 runs for each $(\#estimators, \#features)$, where features used by each estimator are randomly selected. Please, notice that this is not the optimal set of parameters.

Effects On F1-score. As shown in fig. 3, if we fix $\#estimators$, when $\#estimators$ is low (one estimator), F1-score improves considerably as long as $\#features$ increases. When enough $\#estimators$ are used, F1-score stabilizes: we can use less $\#features$ per estimator with almost no effect on F1-score. From fig. 4, we can see that, if we fix $\#features$, F1-score benefits from increasing $\#estimators$. When $\#features$ increases, limiting $\#estimators$ to a low value does not significantly impact the F1-score. In other words, for low $\#features$ it is convenient to run more $\#estimators$: each estimator would work on different subsets of the input features and the union of rules would be hopefully diverse, with a consequent higher F1-score. For the specific case of HEART, we do not notice any significant difference in F1-score by passing from 5 to 20 estimators. However, it is generally convenient to increase $\#estimators$ in order to try as many combinations of features as possible and reduce the variance of results. For datasets with many features, this may make the difference.

Effects On $\#rules$. As shown in fig. 5, if we fix $\#estimators$ and increase $\#features$, $\#rules$ tends to increase up to a certain value, and then stabilizes or get slightly worse. From fig. 6, we notice that, when $\#features$ is low, $\#rules$ tends to increase as long as we increase $\#estimators$. Indeed, the model generates less rules when there are not enough discriminant features; increasing the number of estimators, each estimator discovers different rules that are combined. As long as we increase $\#features$ per estimator, the probability that different estimators work with similar sets of features increases, together with the probability of

generating the same rules (or very similar rules): that's why the size of the rule set tends to stabilize. In this cases, it is convenient to run less estimators to save execution time. In general, increasing the number of estimators considerably reduces the variance of results.

Effects On $\#atoms$. As shown in fig. 7, if we keep $\#estimators$ fixed, $\#atoms$ of the rule set increases as long as the number $\#features$ increases. If we fix $\#features$ (fig. 8), $\#estimators$ does not seem to affect $\#atoms$ significantly. As usual, increasing $\#estimators$ reduces the variance of the results.

Final Remarks. In conclusion, if we want interpretable rule sets, it is better to use few input features per estimator and as many estimators as possible.

E SCALABILITY EVALUATION

Here, we extensively test the scalability of LIBRE. We use up to 50 features and investigate also the impact of class imbalance on the execution time.

Synthetic Dataset. For the scalability evaluation, we synthetically generate a dataset with $1'000'000$ records and 50 continuous features with randomly generated values in the domain $[0, 100]$. Then, we randomly generate four sets of binary labels with a class imbalance ratio of 0.001, 0.01, 0.1, and 0.5 respectively.

Settings. We vary the number of records (10'000, 100'000, 500'000, 1'000'000), features (10, 20, 50), and class imbalance ratio (0.001, 0.01, 0.1, 0.5): for each dataset configuration, LIBRE runs up to 100 times with different randomly generated subsets of features of size 10, 20, and 50; the average execution time in seconds is reported as a sum of two contributions: rule generation and simplification times. Times refer to one weak learner only: if N weak learners run in parallel, the reported time is still a good estimate. Before executing LIBRE, we discretize the dataset with a discretization threshold equal to 6, that we empirically find out to be a good value. The simplification procedure runs on the top 500 rules, if more are generated.

Results. As shown in fig. 9, the execution time is dominated by the rule generation term. Given a class imbalance ratio, execution time increases as long as we increase the number of records and features. The generation time also depends on which features are fed into the model for two main reasons: i) ChiMerge encodes bad predictive features with bigger domains, increasing the search space; ii) the generation procedure will struggle more to generate rules when it runs on features that are not that useful to predict the target class. This explains the high variance in the results. Intuitively, as long as the class imbalance ratio gets close to 0.5, the number of processed records increases, together with the execution time. However, we verified experimentally that this effect is somehow compensated by the higher number of negative records. As already pointed out in the main paper, we run the rule generation procedure up to 50 features just for experimental purposes: for practical applications, if interpretability is a need, it is more convenient to limit the number of features and train a bigger ensemble with more learners in order to generate compact rules in a reasonable time.

F FULL EXPERIMENTS

In this section, we report the full experimental campaign. We use the same methods, training procedure, preprocessing, and evaluation measures as the main paper, but results refer to more datasets (table 2). In all datasets the target class is the positive class.

Data Preprocessing. Before running RBF-SVM, we apply standardization to the input data to get better results. The remaining methods have no benefits from standardization in our experiments. For S-BRL and LIBRE, we apply *ChiMerge* discretization algorithm Kerber (1992) with a discretization threshold in $\{6, 4.6, 4\}$; in BRS, discretization is instead controlled by an internal parameter. In both cases, discretization is optimized during training. The remaining algorithms have no explicit need for discretization. For the methods requiring binarization, we apply *one-hot encoding*, except for LIBRE that uses *inverse one-hot encoding*.

Results. Table 3 reports a comparison between LIBRE and the selected methods in terms of F1-score. Table 4 and table 5 reports the average number of rules and atoms, respectively. We also compare the rule sets leading to the best F1-scores for RIPPER-K, BRS, and S-BRL with a few configurations for LIBRE. In fig. 10, we report the average number of rules and atoms per rule, as a function of the F1-score: points at the bottomrigth side of each plot correspond to compact and high predictive rule sets.

G MORE EXAMPLES OF RULE SETS LEARNED BY LIBRE

In this section, we report additional examples for the medical UCI datasets described in table 2 [1] , for which it might be interesting to understand the relation between input features and the predicted diseases.

[1] Different rule sets may be obtained depending on how folds are randomly built during cross validation.

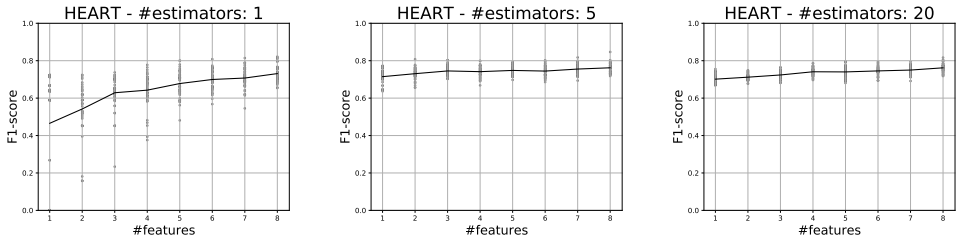


Figure 3: HEART dataset: F1-score as a function of #features for two different values of #estimators.

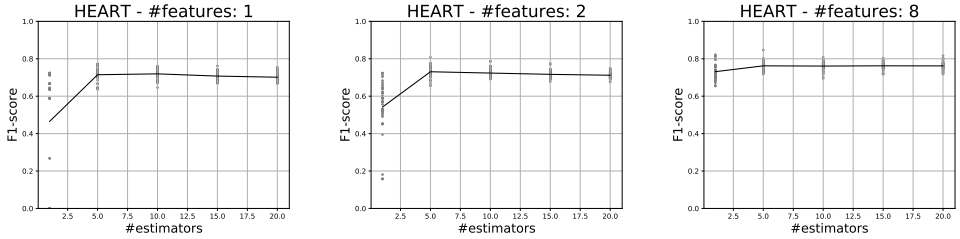


Figure 4: HEART dataset: F1-score as a function of #estimators for four different values of #features.

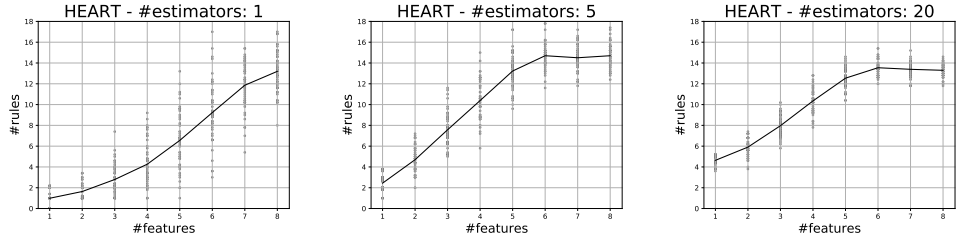


Figure 5: HEART dataset: #rules as a function of #features for four different values of #estimators.

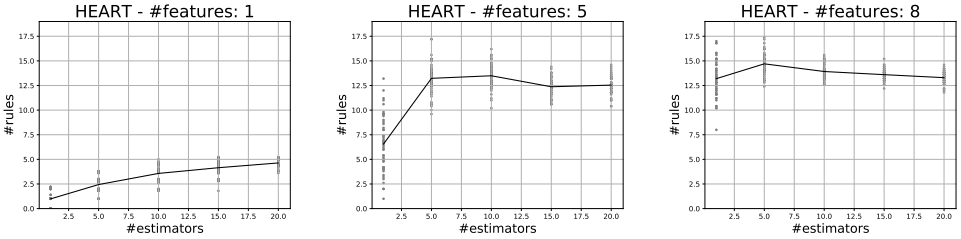


Figure 6: HEART dataset: #rules as a function of #estimators for four different values of #features.

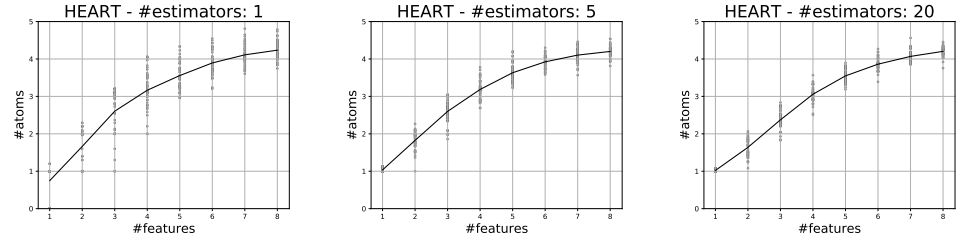


Figure 7: HEART dataset: #atoms as a function of #features for four different values of #estimators.

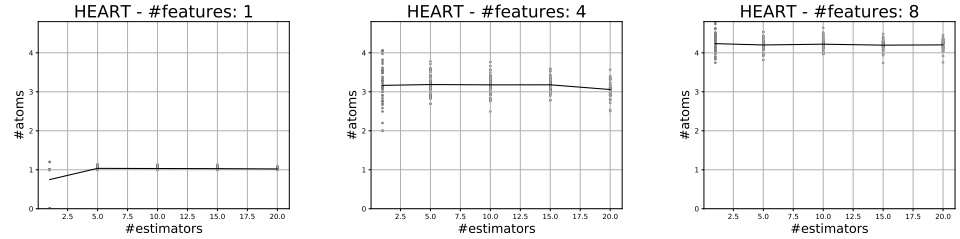


Figure 8: HEART dataset: #atoms as a function of #estimators for four different values of #features.

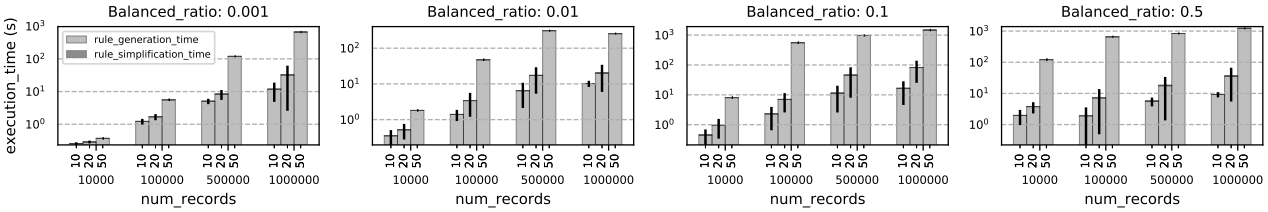


Figure 9: Run time on synthetic data.

Dataset	**#records**	**#features**	**imbalance_ratio**	**target_class**
Adult	48'842	14	.23	>50k
Australian	690	14	.44	2
Balance	625	4	.08	B
Bank	45'211	17	.12	yes
Haberman	306	3	.26	died
Heart	270	13	.51	presence
Ilpd	583	10	.28	liver patient
Liver	345	5	.51	drinks>2
Pima	768	8	.35	1
Sonar	208	60	.53	R
Tictactoe	958	9	.65	positive
Transfusion	748	5	.24	yes
Wisconsin	699	9	.34	malignant
Sap-Clean	287'031	45	.01	crash
Sap-Full	1'554'227	45	.01	crash

Table 2: Characteristics of evaluated datasets.

Dataset	RBF-SVM	RF	DT	RIPPER-K	MODLEM	S-BRL	BRS	LIBRE	LIBRE 3
Adult	.62(.01)	.68(.01)	.68(.01)	.59(.02)	.66(.01)	.68(.01)	.61(.01)	**.70(.01)**	.62(.01)
Australian	.83(.02)	**.86(.02)**	.84(.02)	.85(.02)	.68(.28)	.82(.03)	.83(.03)	.84(.03)	.84(.03)
Balance	.03(.07)	.00(.00)	.01(.03)	.00(.00)	**.16(.04)**	.00(.00)	.00(.00)	**.16(.08)**	.14(.06)
Bank	.46(.01)	.50(.01)	.50(.01)	.44(.04)	.50(.03)	.50(.02)	.32(.05)	**.55(.01)**	.44(.01)
Haberman	.24(.10)	.26(.07)	.36(.08)	.38(.07)	.40(.07)	.17(.21)	.07(.06)	**.41(.04)**	**.41(.04)**
Heart	.78(.06)	**.79(.07)**	.71(.01)	.73(.09)	.39(.31)	.74(.05)	.70(.09)	.77(.06)	.75(.02)
Ilpd	.47(.02)	.44(.08)	.42(.10)	.20(.11)	.48(.08)	.14(.13)	.09(.08)	**.54(.06)**	.52(.04)
Liver	.58(.08)	.58(.07)	.56(.10)	.59(.04)	.58(.07)	.54(.03)	.61(.05)	.60(.07)	**.63(.06)**
Pima	.61(.04)	.63(.04)	.60(.01)	.60(.03)	.38(.18)	.61(.07)	.03(.03)	**.64(.05)**	**.64(.05)**
Sonar	.81(.04)	**.83(.05)**	.75(.05)	.77(.08)	.70(.06)	.76(.05)	.69(.06)	.79(.03)	.76(.04)
Tictactoe	.99(.01)	.99(.01)	.97(.01)	.98(.01)	.55(.10)	.99(.01)	.99(.01)	**1.0(.00)**	.68(.04)
Transfusion	.41(.07)	.35(.06)	.35(.05)	.42(.10)	.42(.08)	.05(.10)	.04(.05)	**.49(.12)**	**.49(.12)**
Wisconsin	**.95(.02)**	**.95(.01)**	.91(.04)	.94(.02)	**.95(.01)**	.94(.02)	.88(.03)	**.95(.01)**	.93(.02)
Sap-Clean	.93(.02)	.93(.01)	.85(.03)	.86(.02)	.88(.01)	.90(.01)	.68(.03)	**.95(.02)**	.72(.03)
Sap-Full	-	-	-	-	-	.81(.02)	-	**.89(.03)**	.68(.04)
Avg Rank	4.0(1.8)	3.1(1.9)	5.5(1.9)	5.3(1.7)	5.0(2.8)	5.3(2.3)	7.3(2.5)	**1.5(0.9)**	4.0(2.6)

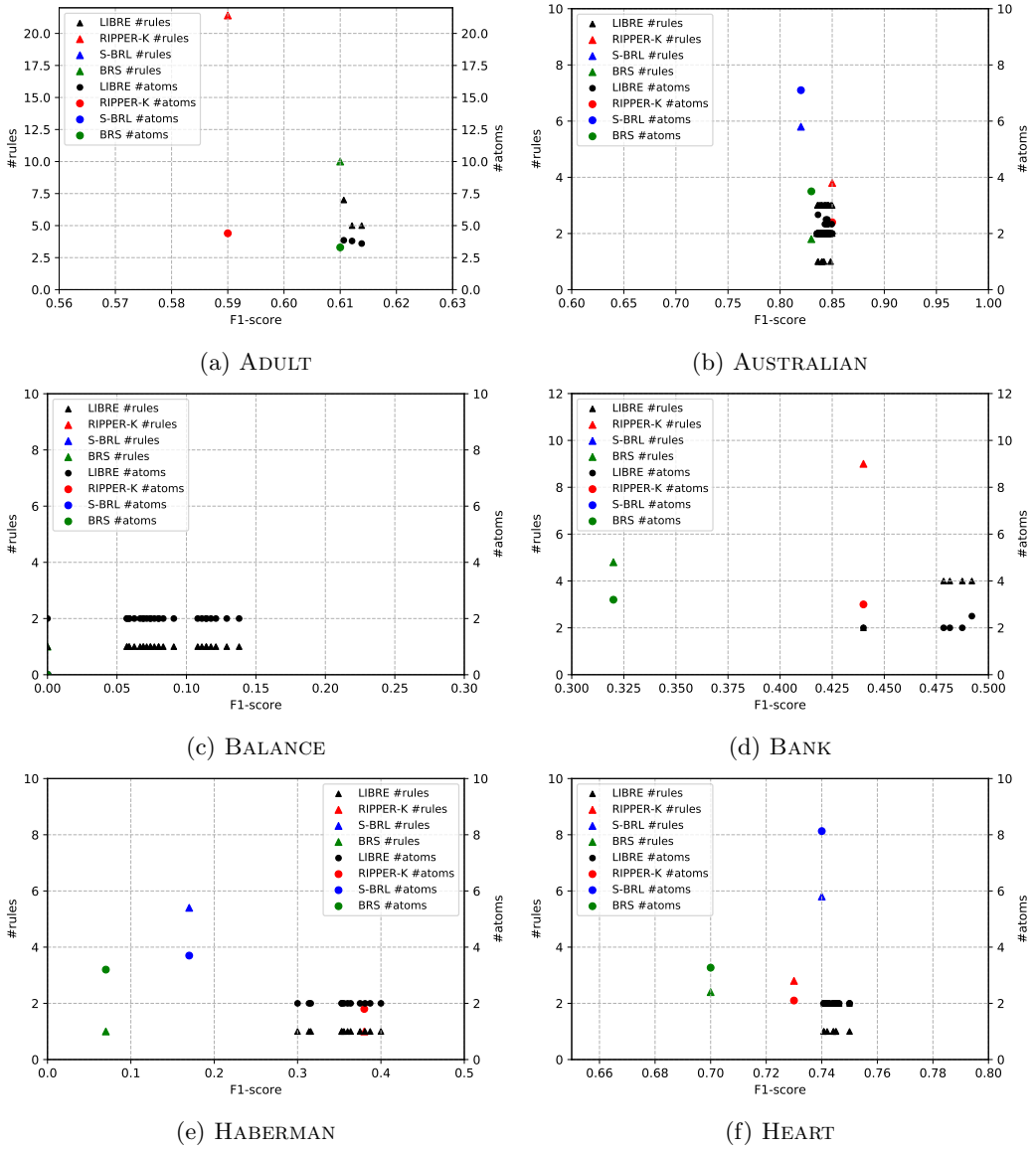
Table 3: F1-score (st. dev. in parenthesis).

Dataset	DT	RIPPER-K	MODLEM	S-BRL	BRS	LIBRE	LIBRE 3
Adult	287.8(6.5)	21.4(5.2)	4957.8(36.3)	71.4(2.1)	10.0(3.3)	14.0(2.1)	**3.0(0.0)**
Australian	4.0(0.0)	3.8(1.2)	86.6(3.2)	5.8(0.7)	**1.8(0.4)**	2.4(1.4)	2.2(0.7)
Balance	48.0(12.5)	0.0(0.0)	76.5(4.6)	0.0(0.0)	**1.0(0.0)**	9.0(3.0)	**1.0(0.0)**
Bank	545.4(18.3)	9.0(1.8)	3722.6(25.5)	61.2(5.5)	4.8(1.2)	15.0(1.1)	**2.0(0.6)**
Haberman	37.4(4.13)	**1.0(0.0)**	73.6(2.9)	5.4(1.9)	**1.0(0.0)**	1.6(0.8)	1.6(0.8)
Heart	45.6(9.1)	2.8(0.7)	50.6(2.9)	5.8(0.7)	**2.4(0.5)**	10.6(3.0)	2.8(0.4)
Ilpd	80.6(30.2)	**1.0(0.6)**	128.2(7.8)	4.8(0.7)	**1.0(0.0)**	4.4(2.3)	2.2(0.4)
Liver	84.4(15.2)	1.4(0.8)	98.4(1.6)	4.0(0.6)	**1.0(0.0)**	3.4(1.9)	2.8(0.4)
Pima	84.8(43.1)	2.4(2.4)	151.8(7.6)	8.4(0.5)	**1.0(0.0)**	1.6(1.0)	1.6(1.0)
Sonar	15.0(9.3)	3.6(1.4)	48.8(1.6)	3.2(0.7)	**1.0(0.0)**	6.6(1.2)	1.1(0.2)
Tictactoe	60.4(5.2)	10.6(1.6)	25.8(1.6)	12.2(1.2)	9.0(1.1)	9.0(1.1)	**3.0(0.0)**
Transfusion	100.2(48.4)	1.8(0.4)	125.8(6.1)	4.4(0.8)	**1.0(0.0)**	1.2(0.4)	1.2(0.4)
Wisconsin	31.4(5.5)	5.0(0.6)	29.2(1.9)	7.0(1.1)	5.0(0.6)	4.2(0.7)	**3.0(0.0)**
Sap-Clean	622.4(51.9)	19.3(3.6)	3944.5(18.8)	47.7(4.4)	20.2(3.5)	13.0(2.4)	**3.0(0.0)**
Sap-Full	-	-	-	56.4(4.6)	-	17.5(5.2)	**3.0(0.0)**
Avg Rank	5.9(0.9)	3.3(1.3)	6.5(0.9)	4.8(0.8)	**1.7(1.1)**	3.1(1.1)	**1.7(0.8)**

Table 4: #rules (st. dev. in parenthesis).

Dataset	DT	RIPPER-K	MODLEM	S-BRL	BRS	LIBRE	LIBRE 3
ADULT	9.3(0.0)	4.4(0.3)	4.3(0.1)	87.0(3.2)	**3.3(0.1)**	7.8(1.0)	6.5(0.7)
AUSTRALIAN	**2.0(0.0)**	2.4(0.3)	2.3(0.1)	7.1(1.0)	3.5(0.3)	4.4(1.8)	4.4(1.3)
BALANCE	4.4(2.9)	0.0(0.0)	3.5(0.0)	0.0(0.0)	4.0(0.0)	**2.1(0.0)**	**2.1(0.0)**
BANK	9.5(0.0)	3.0(0.2)	3.0(0.0)	89.0(7.8)	3.2(0.4)	4.7(0.5)	**2.0(0.1)**
HABERMAN	4.6(3.3)	**1.8(0.4)**	2.2(0.1)	3.7(1.0)	3.2(0.7)	2.1(0.3)	2.1(0.3)
HEART	6.2(0.3)	**2.1(0.3)**	2.3(0.1)	8.1(1.3)	3.3(0.2)	7.7(0.7)	6.1(2.7)
ILPD	8.5(1.6)	**2.1(1.3)**	**2.1(0.0)**	4.4(0.4)	2.8(0.4)	3.3(1.5)	3.0(0.6)
LIVER	8.7(1.0)	**1.3(0.4)**	2.1(0.1)	3.0(0.3)	3.4(0.5)	2.5(1.0)	**1.3(0.4)**
PIMA	6.9(2.5)	2.4(0.5)	**2.1(0.1)**	6.3(0.8)	3.6(0.5)	2.5(0.7)	2.5(0.7)
SONAR	3.8(1.5)	2.1(0.2)	**1.4(0.0)**	8.2(2.1)	4.0(0.0)	3.7(1.0)	2.2(0.8)
TICTACTOE	6.7(0.1)	**2.1(0.2)**	3.5(0.0)	21.8(1.6)	3.5(0.1)	3.8(0.8)	3.0(0.0)
TRANSFUSION	6.9(2.5)	2.8(0.4)	**2.3(0.0)**	3.8(0.6)	3.0(0.6)	2.8(0.7)	2.8(0.7)
WISCONSIN	6.1(0.4)	**2.0(0.2)**	2.2(0.1)	5.9(1.0)	3.3(0.3)	3.2(1.0)	2.8(1.0)
SAP-CLEAN	15.2(1.0)	3.8(0.3)	3.4(0.1)	75.4(4.0)	3.9(0.4)	3.3(0.2)	**3.0(0.1)**
SAP-FULL	-	-	-	85.6(9.7)	-	4.7(0.3)	**4.2(0.2)**
Rank	5.8(1.6)	**2.3(1.4)**	**2.3(1.0)**	6.2(1.0)	4.2(1.3)	3.7(1.5)	2.5(1.7)

Table 5: #atom (st. dev. in parenthesis).

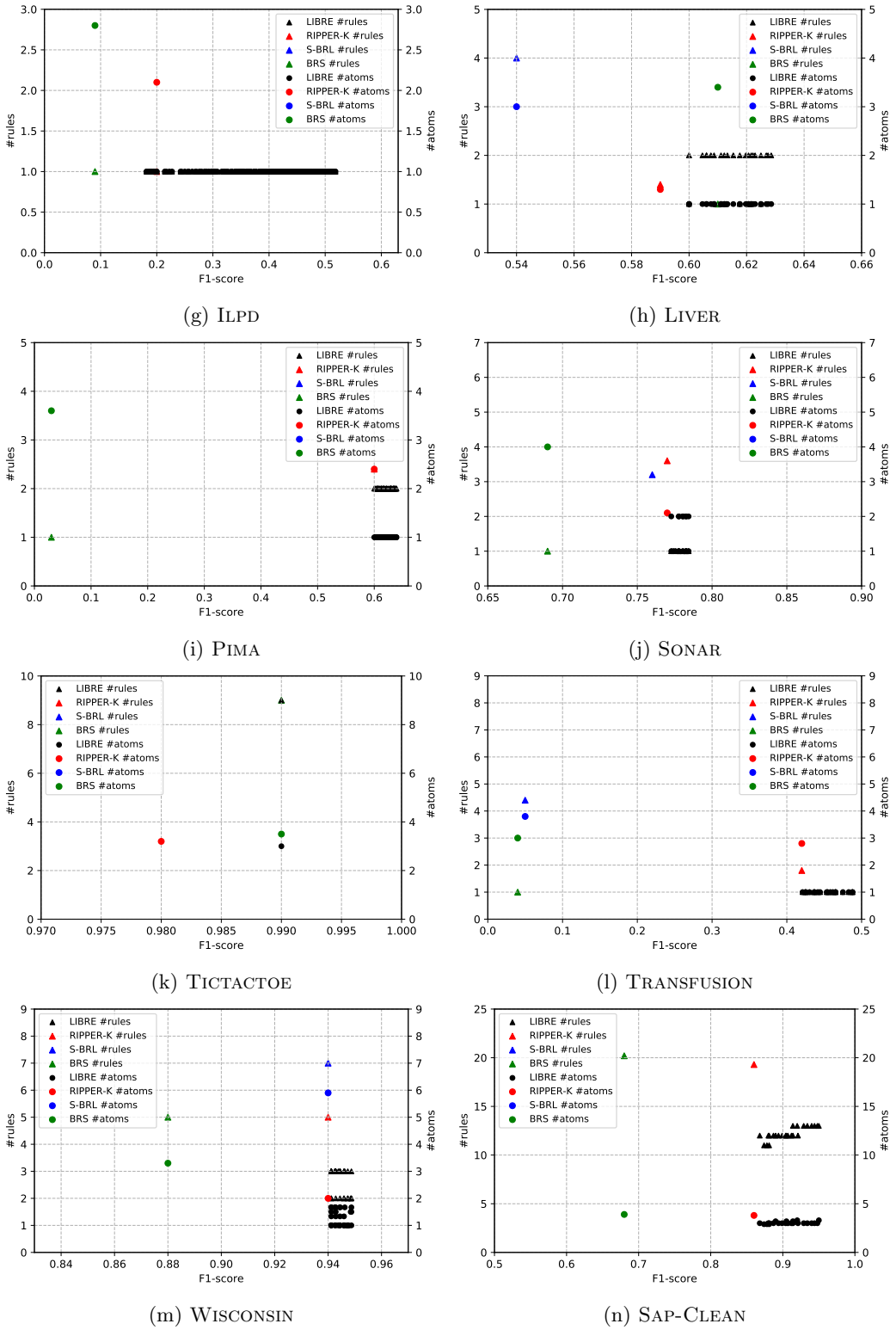


(a) ADULT (b) AUSTRALIAN

(c) BALANCE (d) BANK

(e) HABERMAN (f) HEART

Figure 10: F1-score vs. #rules and #atoms



(g) Ilpd (h) Liver

(i) Pima (j) Sonar

(k) Tictactoe (l) Transfusion

(m) Wisconsin (n) Sap-Clean

Figure 10: F1-score vs. #rules and #atoms

IF (
number_of_positive_axillary_nodes $\in [2, max]$
)
THEN died within 5 years
ELSE survived 5 years or longer

Figure 11: Example of rules learned by LIBRE for HABERMAN.

IF (
slope_of_the_peak_exercise $\in \{flat, downsloping\}$ **AND**
number_of_major_vessels $\in [1, 3]$
)
OR (
chest_pain_type $\in \{asymptomatic\}$ **AND**
thal $\in \{reversable\_defect\}$
)
OR (
sex $\in \{male\}$, **AND**
fasting_blood_sugar_>120mg/dl $\in \{False\}$ **AND**
number_of_major_vessels $\in [1, 3]$
)
THEN class = presence
ELSE class = absence

Figure 12: Example of rules learned by LIBRE for HEART.

IF (
TB $\in [min, 2)$ **AND**
sgbp $\in [min, 42)$)
OR (
TB $\in [min, 2)$ **AND**
alkphos $\in [min, 184)$
)
OR (
age $\in [35, 39), [56, 57)$ **AND**
sgbp $\in [42, 148)$
)
THEN class = liver patient
ELSE class = non liver patient

Figure 13: Example of rules learned by LIBRE for ILPD.

IF (
mean_corpuscular_volume $\in [90, 96)$
)
OR (
gamma_glutamyl_transpeptidase $\in [20, max]$
)
THEN liver_disorder = True
ELSE liver_disorder = False

Figure 14: Example of rules learned by LIBRE for LIVER.

IF (
glucose $\in$ [158, max] **AND**
blood_pressure $\in [56, max]$
)
OR (
glucose $\in$ [110, 158] **AND**
BMI $\in [30.7, max)$
)
OR (
pregnancies $\in$ [4, max] **AND**
diabetes_predigree_func $\in [0.529, max]$
)
THEN diabetes = True
ELSE diabetes = False

Figure 15: Example of rules learned by LIBRE for PIMA.

IF (
months_since_last_donation $\in$ [0, 8) **AND**
total_blood_donated $\in [1250, max)$
)
THEN transfusion = Yes
ELSE transfusion = No

Figure 16: Example of rules learned by LIBRE for TRANSFUSION.

IF (
uniformity_of_cell_shape $\in [5, max]$
)
OR (
clump_thickness $\in [2, max]$ **AND**
bare_nuclei $\in [8, max)$
)
OR (
clump_thickness $\in [7, max]$ **AND**
marginal_adhesion $\in [1, 2), [4, max)$
)
THEN transfusion = Yes
ELSE transfusion = No

Figure 17: Example of rules learned by LIBRE for WISCONSIN.

References

Apache spark. `http://spark.apache.org/`.

R. Kerber. Chimerge: Discretization of numeric attributes. In *Proc. of the 10th Nat. Conf. on Artif. Intel., AAAI*, pages 123–128, 1992.

M. Muselli and A. Quarati. Shadow clustering : A method for monotone boolean function synthesis. 2004.

M. Muselli and A. Quarati. Reconstructing positive boolean functions with shadow clustering. In *Proc. of the 2005 Europ. Conf. on Circuit Theory and Design, ECCTD*, volume 3, pages III/377 – III/380 vol. 3, 2005.