RPA

de Vos, Wout; Balvert, Marleen

Publication date:
2023

Document Version
Early version, also known as pre-print

No. 2023-015

RPA: LEARNING INTERPRETABLE INPUT-OUTPUT RELATIONSHIPS BY COUNTING SAMPLES

By

Wout de Vos, Marleen Balvert

6 July 2023

ISSN 0924-7815
ISSN 2213-9532

Journal of Machine Learning Research 1 (2000) 1-48 Submitted 4/00; Published 10/00

RPA: learning interpretable input-output relationships by counting samples

Wout de Vos W.M.DEVOS@TILBURGUNIVERSITY.EDU
Tilburg University
Department of Econometrics and Operations Research
PO Box 90153, The Netherlands

Marleen Balvert M.BALVERT@TILBURGUNIVERSITY.EDU
Tilburg University
Department of Econometrics and Operations Research
Zero Hunger Lab
PO Box 90153, The Netherlands

Editor:

Abstract

This work proposes a fast solution algorithm to a fundamental data science problem, namely to identify Boolean rules in disjunctive normal form (DNF) that classify samples based on binary features. The algorithm is an explainable machine learning method: it provides an explicit input-output relationship. It is based on hypothesis tests through confidence intervals, where the used test statistic requires nothing more than counting the number of cases and the number of controls that possess a certain feature or a set of features, reflecting the potential AND clauses of the Boolean phrase. Extensive experiments on simulated data demonstrate the algorithm's effectivity and efficiency. The efficiency of the algorithm relies on the fact that the bottleneck operation is a matrix multiplication of the input matrix with itself. More than only a solution algorithm, this paper offers a flexible and transparent theoretical framework with a statistical analysis of the problem and many entry points for future adjustments and improvements. Among other things, this framework allows one to assess the feasibility of identifying the input-output relationships given certain easily-obtained characteristics of the data.

Keywords: Interpretability – Binary classification – Boolean rules in DNF – Confidence intervals – Feasibility analysis.

1. Introduction

Over the recent years the interest in and need for explainable machine learning methods has increased. Interpretability can be considered at various levels: in some cases it is sufficient to know which of the features influence a model's prediction and are hence assumed to be causative of the predicted variable. This information can be obtained from feature importance scores or feature selection methods. In other cases users are not only interested in *which* features explain the dependent variable, but also *how* they influence the prediction. This means that the functional form of the relationship between the independent variables

and the prediction needs to be understandable, which requires the machine learning model itself to be explainable.

When developing an explainable machine learning model, one needs to make assumptions about the functional relationship between the input and the output data. For example in a linear regression the functional form between input and output is assumed to be linear. In this work we focus on Boolean rules in disjunctive normal form (DNF) that constitute a binary classification from binary input data. Boolean rules in DNF classify samples based on a logic rule that consists of an OR combination of AND clauses, for example "IF a sample has characteristics X AND Y, OR if they have characteristics A AND B AND C, OR if they have characteristic Z, then classify as case, else as control".

Deriving Boolean rules in DNF, also termed the Logical Analysis of Data (LAD, Hammer and Bonates (2006)), has been a research topic since Valiant's seminal paper in 1986. Even though Daniely (2016) showed that learning Boolean phrases in DNF is hard, several works have developed efficient algorithms. Many of the earliest algorithms, such as AQ15 (Michalski et al., 1986), FOIL (Quinlan, 1990), RIPPER (Cohen, 1995) and variations thereof, use a sequential covering approach. Later algorithms were based on combinatorial approaches (Hammer and Bonates, 2006) including mixed integer linear programming (MILP) (Dash et al., 2018; Chang et al., 2012; Balvert, 2021). Wu et al. (2011) use an enumerative approach, which has the advantage that the obtained solution is globally optimal. This holds for the use of mixed integer linear programs (MILPs) as well (Knijnenburg et al., 2016; Chang et al., 2012; Malioutov and Varshney, 2013; Dash et al., 2018; Balvert, 2021). Nevertheless, enumerative approaches and MILPs run into computational challenges when the number of samples or features grows. This is why Malioutov and Varshney (2013), Dash et al. (2018) and Balvert (2021) have developed MILP-based heuristic algorithms to speed up the computational process. Their methods can handle datasets with up to thousands (Dash et al., 2018) or even 10,000 (Balvert, 2021) samples and features. As these methods are heuristics, there is a trade-off between computational complexity and classification accuracy (Balvert, 2021). Also metaheuristics such as genetic algorithms have been used to abstract Boolean phrases in DNF from data (McCallum and Spackman, 1990).

Some machine learning approaches make use of Boolean phrases in DNF without deriving the explicit Boolean phrase as the sole means for classification. Decision trees can be formulated such that they follow a DNF. Seyedhosseini and Tasdizen (2015) developed such disjunctive normal decision trees and the corresponding disjunctive normal random forests. While the DNF is used at this method's core, as any random forest this method does not output an interpretable description of the input-output relationship. Logic regression (Ruczinski et al., 2003) constructs several Boolean expressions - not restricted to DNF - and takes a linear combination of a set of Boolean expressions to reach a prediction. This approach is suitable for regression tasks rather than classification tasks.

In this work we propose a fast algorithm to abstract Boolean rules in DNF from binary data based on hypothesis tests through confidence domains. We call it the *Remaining Positives Algorithm* (RPA), after the Remaining Positives statistics which are the core of the algorithm. These test statistics require nothing more than counting the number of cases and the number of controls that possess a certain feature or a set of features, reflecting potential AND clauses of the Boolean phrase. In each iteration, one AND clause is identified and the corresponding samples and features are removed from the dataset, allowing the algorithm

to find an AND clause that fits best for the remaining samples in the next iteration. Due to its simplicity, - the bottleneck operation is matrix multiplication of the input matrix with its own transpose - the algorithm can handle large amounts of data: we have achieved good results in experiments with datasets containing up to 10,000 samples and 2,500 features, in a fraction of the time required by existing methods (Balvert, 2021).

Most developed classifiers work very well for datasets of limited size, but cannot keep up with the steady increase in the number of samples and features that datasets nowadays contain. Earlier approaches suffer computationally from an increase in the number of samples, as this increases the model's complexity - e.g. the number of binary variables in the MILPs of Dash et al. (2018) and Balvert (2021). From a statistical point of view however an increased number of samples leads to inceased statistical power and is hence desirable. The computational complexity of the approach proposed in this work suffers only linearly from an increased number of samples, enabling the method to make use of the increased statistical power obtained with an increased sample size.

In addition to proposing a new algorithm, our second contribution constitutes a theoretical framework on how to assess the model's ability to identify Boolean rules in DNF of various sizes in datasets with various characteristics. We propose a metric that we refer to as the "degree of overlap", which is a measure for the distinguishability between AND clauses contained in the Boolean rule and any other AND clauses that are not contained in the Boolean rule. We use the degree of overlap to determine how likely it is that the algorithm is able to retrieve the AND clauses of the Boolean rule for a given dataset. The degree of overlap depends only on the number of samples, cases and features in a dataset as well as the frequency of occurrence of the features, allowing for analyzing the effect of these parameters on the performance of the algorithm. More generally, as the proposed algorithm is based on fundamental statistics, the degree of overlap also provides an indication on how many samples one would need in order to be able to find Boolean rules in DNF of a certain complexity that explain the class label from the independent input data.

2. Methods

We explain the Remaining Positives Algorithm (RPA) using the following structure. Section 2.1 formulates the problem. Sections 2.2, 2.3 and 2.4 develop the techniques that are used in the algorithm. Section 2.5 integrates the techniques into the solution algorithm.

2.1 Problem Formulation

We are given a dataset (X, y) with a binary $N \times P$ matrix X and a binary $N \times 1$ vector y that depends on the matrix X via an unknown boolean rule and error. The goal is to find the boolean rule.

Each row $i \in [N]$ of X corresponds to a sample and each column $j \in [P]$ of X corresponds to a feature. We assume that the matrix entries are independently Bernoulli distributed as $X_{ij} \overset{iid}{\sim} Ber(p)$ with a common probability $p \in [0, 1]$.

The vector y depends on the matrix X via a boolean rule in disjunctive normal form (DNF), which is an OR-combination of AND-clauses. We represent such a rule as a set of clauses, where each clause is a set of features. The size of a clause is the number of features

it contains. An example of a rule with three clauses of sizes 1, 2 and 4 is given by

$$rule = \big\{\{1\}, \{101, 102\}, \{201, 202, 203, 204\}\big\} \, . \tag{1}$$

This translates to the decision rule "IF sample i has $X_{i,1} = 1$, OR if they have $X_{i,101} = 1$ AND $X_{i,102} = 1$, OR if they have $X_{i,201} = 1$ AND $X_{i,202} = 1$ AND $X_{i,203} = 1$ AND $X_{i,204} = 1$, then $y_i = 1$, else $y_i = 0$". We say that a clause is *active* for a sample i if all entries X_{ij} corresponding to the features j of the clause are equal to 1. For instance, for sample i, the clause $c = \{101, 102\}$ is active if $X_{i,101} = 1$ and $X_{i,102} = 1$. The boolean rule applied to sample i determines that $y_i = 1$ if at least one of the clauses of the rule is active, and if the sample is not perturbed by error.

The error is modelled as an independent Bernoulli random variable $\epsilon_i \overset{iid}{\sim} Ber(\beta)$ for every sample i, with a common error probability $\beta \in [0, 1]$. If $\epsilon_i = 1$, then the observed y_i is the reverse outcome of the boolean rule applied to sample i. Hence, if we denote the XOR-operator by $\oplus$, the dependent variable y_i for a dataset with boolean rule (1) is given by

$$y_i = \Big[(X_{i,1}) \vee (X_{i,101} \wedge X_{i,102}) \vee (X_{i,201} \wedge X_{i,202} \wedge X_{i,203} \wedge X_{i,204})\Big] \oplus \epsilon_i \, ,$$

or, more generally, y_i is given by

$$y_i = \Big(\bigvee_{c \, \in \, rule} \; \bigwedge_{j \, \in \, c} X_{ij} \Big) \oplus \epsilon_i \, .$$

To develop intuition for the problem and illustrate concepts, we consider the following example dataset, which we will revisit throughout the paper. The dataset has $N = 1,500$ samples and $P = 1,000$ features, it contains the rule given in (1) and it is generated with $p = 0.5$ and $\beta = 0.05$. Table 1 is a stylized representation of the example dataset. It contains a subset of 25 samples and 17 features, where the displayed samples have been selected uniformly. The columns have been reorganized such that the features contained in the rule come first. Also, the rows have been reorganized such that rows where the clauses are active come first. The 1s that make one of the clauses of the rule active are colored blue if the corresponding sample is a case ($y_i = 1$), or red if the corresponding sample is an erroneous control ($y_i = 0$).

An important phenomenon that can be observed in Table 1 is that the clause $\{1\}$ of size 1 is active in approximately twice as many rows as the clause $\{101, 102\}$ of size 2 and in approximately eight times as many rows as the clause $\{201, 202, 203, 204\}$ of size 4. It can be verified that this is a straightforward consequence of the definition of the rule and the fact that $p = 0.5$.

1	101	102	201	202	203	204	990	991	992	993	994	995	996	997	998	1000	y
1	0	1	1	0	1	0	0	1	0	1	0	1	1	0	0	1	1
1	1	1	1	0	0	1	1	1	0	1	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	0	1
1	1	0	0	1	0	1	0	0	1	1	0	1	1	1	1	0	1
1	1	1	0	1	0	1	1	0	1	0	0	1	0	0	1	1	1
1	0	0	0	0	1	1	1	0	1	0	1	1	1	0	0	1	1
1	0	1	0	0	1	1	1	1	1	0	0	0	0	1	1	1	1
1	0	1	1	0	1	0	1	0	1	1	1	1	1	0	1	0	1
1	0	0	0	1	1	1	0	1	1	1	1	1	1	1	1	0	1
1	1	1	0	1	1	0	1	1	1	1	0	1	0	1	0	0	1
1	1	0	1	0	1	0	1	0	0	0	1	0	1	1	1	1	1
0	1	1	1	0	1	0	0	1	0	0	0	0	0	1	1	1	1
0	1	1	0	0	1	0	0	0	1	0	0	1	1	1	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0	1	1	0	0	0	1
0	0	0	1	1	1	1	0	1	1	0	0	0	1	0	0	1	1
1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	1	0
0	1	0	1	0	1	0	1	0	1	1	1	0	1	1	1	1	0
0	0	1	0	1	1	0	1	1	0	0	1	0	0	0	0	1	0
0	0	0	1	0	1	1	0	1	0	1	1	0	1	1	1	1	0
0	0	1	0	0	0	1	1	1	1	1	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0	1	1	1	1	0	0	0	0	1	0
0	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0
0	0	0	1	0	0	0	1	1	0	1	0	1	1	0	0	0	0
0	0	0	1	1	0	1	1	0	0	1	1	1	1	1	1	1	0

Table 1: A selection of rows and columns from the example dataset.

Finally, we introduce some notation that we use in later sections. We denote by $\mathcal{C}^s(X)$ the set of all clauses of size s. For instance, in case of the example dataset with $P = 1,000$ features, the sets of all clauses of sizes 1 and 2 are given by

$$\mathcal{C}^1(X) = \Big\{\{1\},\{2\},\{3\},\ldots,\{1000\}\Big\}\,,$$
$$\mathcal{C}^2(X) = \Big\{\{1,2\},\{1,3\},\{1,4\},\ldots,\{999,1000\}\Big\}\,.$$

Oserve that for a size s, the set of all clauses has cardinality $|\mathcal{C}^s(X)| = \binom{P}{s}$.

We distinguish two types of clauses. Clauses that are contained in the rule are called *generating*, because they play a role in the realization of the vector y. The set of generating clauses of a dataset (X, y) is denoted by $\mathcal{G}(X)$. Sometimes we include a superscript, i.e., $\mathcal{G}^s(X)$, to specify the set of generating clauses of size s. Clauses that are not contained in the rule are called *dormant*. Similarly, the set of all dormant clauses is denoted by $\mathcal{D}(X)$, and $\mathcal{D}^s(X)$ denotes all dormant clauses of size s.

With this new vocabulary, we can formulate the problem as being given a dataset (X, y) and having to find the set of generating clauses $\mathcal{G}(X)$.

2.2 Finding Generating Clauses of Size 1

First, we develop a classification procedure that labels a clause $c \in \mathcal{C}^1(X)$ of size 1 as generating, dormant, or ambiguous. Then, we find generating clauses by classifying all clauses of size 1. Since the set $\mathcal{C}^1(X)$ is simply the set of individual features, it is computationally feasible to classify all clauses of size 1. The classification procedure of a clause is based on two statistics, $RP_1(c)$ and $RP_0(c)$, that will be introduced next.

2.2.1 RP-statistics for Clauses of Size 1

The *Remaining Positives (RP)* statistics count for a given clause $c \in \mathcal{C}^1(X)$ of size 1 the number of cases where c is active, referred to as $RP_1(c)$, and the number of controls where c is active, referred to as $RP_0(c)$.

Definition 1 (RP-statistics for clauses of degree 1) *For a clause $c \in \mathcal{C}^1(X)$ of size 1, the Remaining Positives statistics are given by*

$$RP_1(c) = \sum_{i \in [N]} \mathbb{1}\{X_{ic} = 1 \wedge y_i = 1\} \ ,$$
$$RP_0(c) = \sum_{i \in [N]} \mathbb{1}\{X_{ic} = 1 \wedge y_i = 0\} \ .$$

The key insight of this paper is that generating clauses often have significantly higher RP_1 values and significantly lower RP_0 values than dormant clauses. The RP-statistics count samples. What distinguishes generating and dormant clauses are the probabilities that a given sample i is counted by $RP_1(c)$ or $RP_0(c)$. We compute these probabilities and derive the distributions of $RP_1(c)$ and $RP_0(c)$ for generating and dormant clauses separately.

For a generating clause $c \in \mathcal{G}^1(X)$, we consider a sample $i \in [N]$ and compute the probability that $X_{ic} = 1$ and $y_i = 1$. Since c is generating and has size 1, the event $(X_{ic} = 1)$ is sufficient to activate the boolean rule for this sample i. If the error term ϵ_i is not active for this sample, then we indeed observe $y_i = 1$. Hence, the probability that sample i is counted by $RP_1(c)$ is

$$\begin{aligned}\mathbb{P}[X_{ic} = 1 \wedge y_i = 1] &= \mathbb{P}[X_{ic} = 1] \cdot \mathbb{P}[y_i = 1 | X_{ic} = 1] \\ &= \mathbb{P}[X_{ic} = 1] \cdot \mathbb{P}[\epsilon_i = 0] \\ &= p \cdot (1 - \beta) \ .\end{aligned}$$

Similarly, for a generating clause $c \in \mathcal{G}^1(X)$, the probability that a row $i \in [N]$ is counted by $RP_0(c)$ is

$$\begin{aligned}\mathbb{P}[X_{ic} = 1 \wedge y_i = 0] &= \mathbb{P}[X_{ic} = 1] \cdot \mathbb{P}[y_i = 0 | X_{ic} = 1] \\ &= \mathbb{P}[X_{ic} = 1] \cdot \mathbb{P}[\epsilon_i = 1] \\ &= p \cdot \beta \ .\end{aligned}$$

Since $RP_1(c)$ and $RP_0(c)$ sum over N *iid* Bernoulli random variables, they are binomially distributed as follows.

Proposition 2 (RP-distributions of generating clauses of size 1) *Consider a generating clause $c \in \mathcal{G}^1(X)$ of size 1. The distributions of $RP_1(c)$ and $RP_0(c)$ are given by*

$$RP_1(c) \sim Bin(N, p \cdot (1 - \beta)),$$
$$RP_0(c) \sim Bin(N, p \cdot \beta),$$

with cumulative density functions G_1 and G_0, respectively.

Now we derive the RP-distributions for a dormant clause $c \in \mathcal{D}^1(X)$. Since c is dormant, whether a sample i is a case or a control is independent of the entry X_{ic}. Hence, we interpret $RP_1(c)$ as iterating over the cases and counting those samples i where $X_{ic} = 1$. Similarly, we interpret $RP_0(c)$ as counting the number of controls where $X_{ic} = 1$. Since $RP_1(c)$ and $RP_0(c)$ sum over *iid* Bernoulli random variables, they are binomially distributed as follows, where N_0 and N_1 denote the number of controls and cases, respectively.

Proposition 3 (RP-distributions of dormant clauses of size 1) *Consider a dormant clause $c \in \mathcal{D}^1(X)$ of size 1. The distributions of $RP_1(c)$ and $RP_0(c)$ are given by*

$$RP_1(c) \sim Bin(N_1, p),$$
$$RP_0(c) \sim Bin(N_0, p),$$

with cumulative density functions D_1 and D_0, respectively.

Often, the RP-distributions for generating and dormant clauses are well-separated. Specifically, the values of RP_1 that generating clauses can attain with nonnegligible probability are often significantly higher than those of dormant clauses. Vice versa, the values of RP_0 of generating clauses tend to be considerably lower than those of dormant clauses. We illustrate this with the example dataset.

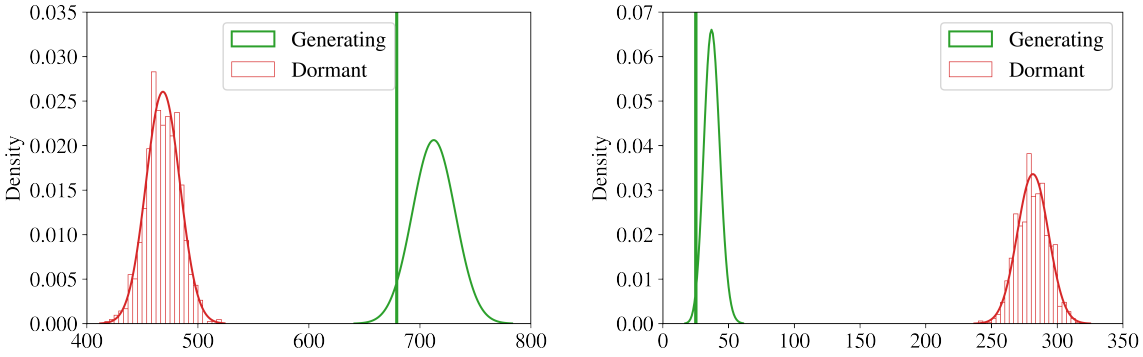
Example 1 *Consider the example dataset, which, we recall, has $N = 1,500$ rows, $P = 1,000$ columns and is generated with $p = 0.5$ and $\beta = 0.05$. The (hidden) boolean rule is $\{\{1\}, \{101, 102\}, \{201, 202, 203, 204\}\}$. The RP-distributions for the generating clause $\{1\}$ of size 1 are given by*

$$RP_1(\{1\}) \sim Bin(1500, 0.475),$$
$$RP_0(\{1\}) \sim Bin(1500, 0.025),$$

and the RP-distributions of dormant clauses $c \in \mathcal{D}^1(X)$ of size 1 are given by

$$RP_1(c) \sim Bin(937, 0.5),$$
$$RP_0(c) \sim Bin(563, 0.5).$$

In Figure 1 the red and green curves represent the RP-distributions and the histograms represent the observed RP-values of the example dataset, for the dormant (red) and generating (green) clauses. Note that there is only one generating clause of size 1, so that the fat green lines represent the observed RP-values for the generating clause. Observe that the RP-values of dormant and generating clauses are well-separated.



(a) RP_1 (b) RP_0

Figure 1: The RP-distributions and the observed RP-values for all clauses of size 1 of the example dataset.

2.2.2 Classification

We classify a given clause of size 1 $c \in \mathcal{C}^1(X)$ by computing $RP_1(c)$ and $RP_0(c)$ and comparing their values with four carefully constructed intervals. We call the interval $\Gamma_1(X)$ the *generating domain* of the statistic $RP_1(c)$. It will be constructed such that for any generating clause $c \in \mathcal{G}^1(X)$, the observed value $RP_1(c)$ lies almost certainly in the interval $\Gamma_1(X)$. Here, the subscript "1" indicates that we refer to the statistic $RP_1(c)$. Similarly, the generating domain $\Gamma_0(X)$ will be constructed such that it almost certainly includes the observed value $RP_0(c)$ of any generating clause $c \in \mathcal{G}^1(X)$. Likewise, the dormant domains $\Delta_1(X)$ and $\Delta_0(X)$ will be constructed such that they almost certainly include the observed $RP_1(c)$ and $RP_0(c)$, respectively, of any dormant clause $c \in \mathcal{D}^1(X)$. Specifically, the intervals will be constructed such that we have, almost certainly,

$$\begin{aligned} c \in \mathcal{G} \implies \quad & RP_1(c) \in \Gamma_1(X) \quad \text{and} \quad RP_0(c) \in \Gamma_0(X) \,, \\ c \in \mathcal{D} \implies \quad & RP_1(c) \in \Delta_1(X) \quad \text{and} \quad RP_0(c) \in \Delta_0(X) \,. \end{aligned}$$

We classify a clause $c \in \mathcal{C}^1(X)$ as *generating* if the values $RP_1(c)$ and $RP_0(c)$ lie in the generating domains, and if at least one of them lies outside its respective dormant domain. The latter allows us to conclude that c is almost certainly not dormant. Specifically, c is labelled as generating if the following two conditions hold.

1. $RP_1(c) \in \Gamma_1(X)$ and $RP_0(c) \in \Gamma_0(X)$;
2. $RP_1(c) \notin \Delta_1(X)$ or $RP_0(c) \notin \Delta_0(X)$.

We classify a clause $c \in \mathcal{C}^1(X)$ as *dormant* if one of the values $RP_1(c)$ and $RP_0(c)$ lies outside its respective generating domain, so that we conclude that c is not generating. Hence, c is labelled as dormant if the following condition holds.

1. $RP_1(c) \notin \Gamma_1(X)$ or $RP_0(c) \notin \Gamma_0(X)$.

In the remaining cases, we classify a given clause $c \in \mathcal{C}^1(X)$ as *ambiguous*. If $RP_1(c)$ and $RP_0(c)$ lie in both their respective generating and dormant domains, we will store c as a

promising candidate for being a generating clause, but we do not exclude that it is dormant. Specifically, c is labelled as ambiguous if the following condition holds.

1. $RP_1(c) \in \Gamma_1(X) \cap \Delta_1(X)$ and $RP_0(c) \in \Gamma_0(X) \cap \Delta_0(X)$.

We implement the classification procedure as follows.

Algorithm 1: Classification of a clause of size 1

Input: Dataset (X, y), clause $c \in \mathcal{C}^1(X)$, probabilities p and β;
Output: Classification of c;
Compute $RP_1(c)$ and $RP_0(c)$;
Compute the intervals $\Gamma_1(X), \Gamma_0(X), \Delta_1(X), \Delta_0(X)$;
if $RP_1(c) \notin \Gamma_1(X)$ **or** $RP_0(c) \notin \Gamma_0(X)$ **then**
  Return *Dormant*;
else if $RP_1(c) \notin \Delta_1(X)$ **or** $RP_0(c) \notin \Delta_0(X)$ **then**
  Return *Generating*;
else
  Return *Ambiguous*.

Now we construct the generating and dormant domains. The generating domains $\Gamma_1(X)$ and $\Gamma_0(X)$ are constructed using the cumulative distribution functions G_1 and G_0 for generating clauses. We recall that values of $RP_1(c)$ are generally higher for generating clauses than for dormant clauses. Hence, we will reserve the lower end of the real numbers $\mathbb{R}_+$ for $\Delta_1(X)$ and the upper end for $\Gamma_1(X)$. We define the generating domain $\Gamma_1(X)$ as the range of values that include the value $RP_1(c)$ of a given generating clause $c \in \mathcal{G}^1(X)$ with probability 0.999, i.e.,

$$\Gamma_1(X) = \left[(G_1)^{-1}(0.001), \infty\right) .$$

Similarly, we recall that values of $RP_0(c)$ are generally lower for generating clauses than for dormant clauses. We define the generating domain $\Gamma_0(X)$ as the lower end of the real numbers $\mathbb{R}_+$ that contains $RP_0(c)$ for a given generating clause $c \in \mathcal{G}^1(X)$ with overwhelming probability, i.e.,

$$\Gamma_0(X) = \left[0, (G_0)^{-1}(0.999)\right] .$$

To construct the dormant domains $\Delta_1(X)$ and $\Delta_0(X)$ we also use the cumulative distribution functions D_1 and D_0, but we first observe an additional subtlety. To construct the intervals such that they almost certainly include $RP_1(c)$ and $RP_0(c)$ for a given dormant clause $c \in \mathcal{D}^1(X)$, we need to observe that the set of dormant clauses is big, namely, approximately of size

$$|\mathcal{D}^1(X)| \approx |\mathcal{C}^1(X)| = \binom{P}{1} = P .$$

Suppose that we constructed the dormant domains $\Delta_1(X)$ and $\Delta_0(X)$ in a similar fashion as the generating domains, so that they include $RP_1(c)$ and $RP_0(c)$ for a given $c \in \mathcal{D}^1(X)$ with probability 0.999. Then, for datasets with a large number of features P, we can expect that for a fair number of dormant clauses the observed values $RP_1(c)$ or $RP_0(c)$ are

not included in the dormant domains $\Delta_1(X)$ and $\Delta_0(X)$. Therefore, we construct $\Delta_1(X)$ so that the expected number of dormant clauses in the dataset with a value $RP_1(c)$ that exceeds the interval $\Delta_1(X)$ is low. For this, we introduce the auxiliary function δ_1, which approximates for a given $x \in \mathbb{R}$ the expected number of dormant clauses $c \in \mathcal{D}^1(X)$ that satisfy $RP_1(c) \geq x$. It is given by

$$\begin{aligned}\delta_1(x) &= \mathbb{E}\Big[\ \Big|\{c \in \mathcal{D}^1(X) :\ RP_1(c) \geq x\}\Big|\ \Big] \\ &= \mathbb{P}\Big[RP_1(c) \geq x \mid c \in \mathcal{D}^1(X)\Big] \cdot \Big|\mathcal{D}^1(X)\Big| \\ &\approx \big(1 - D_1(x)\big) \cdot P\ .\end{aligned}$$

To aim that $\Delta_1(X)$ does not overlap with $\Gamma_1(X)$, we define the dormant domain $\Delta_1(X)$ as the lower end of $\mathbb{R}_+$ such that, expectedly, only one dormant clause $c \in \mathcal{D}^1(X)$ has a value $RP_1(c)$ that exceeds this domain:

$$\Delta_1(X) = \big[0, (\delta_1)^{-1}(1)\big]\ .$$

Similarly, we define the dormant domain $\Delta_0(X)$ such that, expectedly, only one dormant clause has a value $RP_0(c)$ that is too low to be included in the domain. For this, we define the auxiliary function δ_0 as

$$\delta_0(x) = D_0(x) \cdot P\ .$$

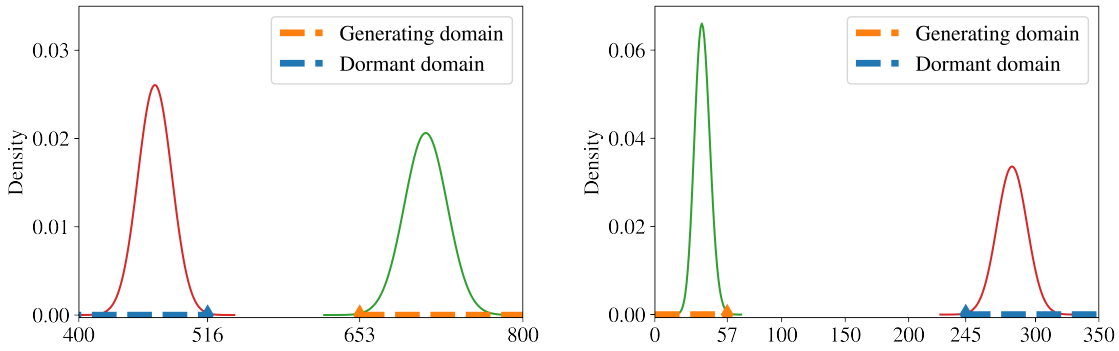
Then, the dormant domain $\Delta_0(X)$ is given by

$$\Delta_0(X) = \big[(\delta_0)^{-1}(1), \infty\big)\ .$$

Example 2 *Reconsider the example dataset (as in Example 1). The domains are given by*

$$\begin{aligned}\Gamma_1(X) &= [653, \infty), \quad \Gamma_0(X) = [0, 57], \\ \Delta_1(X) &= [0, 516], \quad \Delta_0(X) = [245, \infty).\end{aligned}$$

Figure 2, shows the distributions and domains of generating and dormant clauses of size 1.



(a) RP_1 (b) RP_0

Figure 2: The RP-distributions and domains for the dormant and generating clauses of size 1 of the example dataset.

To find generating clauses of size 1, we compute $RP_1(c)$ and $RP_0(c)$ for all clauses of size 1 and apply the classification algorithm. We use the labels 'G' for generating, 'D' for dormant, and 'A' for ambiguous, see Table 2.

$c \in \mathcal{C}^1(X)$	{1}	{2}	{3}	{4}	{5}	...	{998}	{999}	{1000}
$RP_1(c)$	679	468	442	444	473	...	456	443	484
$RP_0(c)$	25	299	264	277	256	...	268	288	297
$Class(c)$	G	D	D	D	D	...	D	D	D

Table 2: RP_1, RP_0 and classifications of a subset of size-1 clauses for the example dataset.

Observe that clause $c = \{1\}$ is correctly classified as generating, due to $RP_1(\{1\}) = 679 \in \Gamma_1(X)$ and $RP_0(\{1\}) = 25 \in \Gamma_0(X)$ and $RP_1(\{1\}) \notin \Delta_1(X)$. All other clauses $c \neq \{1\}$ of size 1 are correctly classified as dormant, since they satisfy at least one of the conditions $RP_1(c) \notin \Gamma_1(X)$ or $RP_0(c) \notin \Gamma_0(X)$.

2.3 Finding Generating Clauses of Size $s \geq 2$

We find generating clauses of size $s \geq 2$ in a similar way as generating clauses of size 1: with RP-statistics and a classification procedure. For clauses of size 2, we compute $RP_1(c)$ and $RP_0(c)$ and apply the classification procedure to all clauses $c \in \mathcal{C}^2(X)$. We will demonstrate that this is computationally feasible. For clauses of size $s \geq 3$ this is not computationally feasible. Therefore, we introduce a search heuristic to find promising clauses of sizes $s \geq 3$ and classify only these promising clauses.

In this section, when searching for a generating clause of $s \geq 3$ we assume that we are given a dataset (X, y) that has no generating clauses of size strictly smaller than s. If the dataset contained a generating clause of size $1, ..., s-1$, we assume it has been found and eliminated using the procedure that will be explained in Section 2.4. This is a necessary assumption, as the presence of a clause of size 1 reduces the visibility of a clause of size 2. This phenomenon will be further explained in Section 2.4.

2.3.1 RP-statistics & Classification for clauses of size $s \geq 2$

For a clause $c = \{c_1, ..., c_s\} \in \mathcal{C}^s(X)$ of size $s \geq 2$ with features $c_1, ..., c_s \in [P]$, the statistic $RP_1(c)$ counts the number of cases where c is active, and $RP_0(c)$ counts the number of controls where c is active. The qualifier "remaining" in the name "remaining positives" reflects that many of the rows that were counted by the size-1 statistics $RP_1(c_1), ..., RP_1(c_s)$ and $RP_0(c_1), ..., RP_0(c_s)$ are not counted by the statistics and $RP_1(c)$ and $RP_0(c)$. We could imagine that features $c_1, ..., c_s$ are merged into a "feature of size s" where a 1 remains in rows where all features $c_1, ..., c_s$ have a 1.

Definition 4 (RP-statistics for clauses of size s) *Consider a clause $c = \{c_1, ..., c_s\} \in \mathcal{C}^s$ of size $s \in \mathbb{N}$, where $c_1, ..., c_s \in [P]$ are features. The "Remaining Positives"-statistics of*

c are given by

$$RP_1(c) = \sum_{i\in[N]} \mathbb{1}\{ \bigwedge_{j\in c} X_{ij} = 1 \ \wedge y_i = 1\} \ ,$$
$$RP_0(c) = \sum_{i\in[N]} \mathbb{1}\{ \bigwedge_{j\in c} X_{ij} = 1 \ \wedge y_i = 0\} \ .$$

Following a similar logic as for clauses of size 1, the distributions of the RP-statistics for clauses of general size are as follows.

Proposition 5 (Distributions of the RP-statistics of clauses of size s) *For a generating clause $c \in \mathcal{G}^s(X)$ of size $s \in \mathbb{N}$, the distributions of statistics $RP_1(c)$ and $RP_0(c)$ are given by*

$$RP_1(c) \sim Bin(N, p^s \cdot (1-\beta)),$$
$$RP_0(c) \sim Bin(N, p^s \cdot \beta),$$

with cumulative density functions G_1^d and G_0^d, respectively.

For a dormant clause $c \in \mathcal{D}^s(X)$ of size $s \in \mathbb{N}$, the distributions of $RP_1(c)$ and $RP_0(c)$ are given by

$$RP_1(c) \sim Bin(N_1, p^s),$$
$$RP_0(c) \sim Bin(N_0, p^s),$$

with cumulative density functions D_1^s and D_0^s, respectively.

To classify a given clause $c \in \mathcal{C}^s$ of size $s \in \mathbb{N}$, we compute the generating domains as

$$\Gamma_1^s(X) = \big[(G_1^s)^{-1}(0.001), \infty\big),$$
$$\Gamma_0^s(X) = \big[0, (G_0^s)^{-1}(0.999)\big].$$

Additionally, we compute the dormant domains using the auxiliary functions

$$\delta_1^s(x) = \big(1 - D_1^s(x)\big) \cdot \binom{P}{s},$$
$$\delta_0^s(x) = D_0^s(x) \cdot \binom{P}{s}.$$

The dormant domains are then given by

$$\Delta_1^s(X) = \big[0, (\delta_1^s)^{-1}(1)\big],$$
$$\Delta_0^s(X) = \big[(\delta_0^s)^{-1}(1), \infty\big).$$

We implement the classification procedure for clauses of size s as follows.

Algorithm 2: Classification of clauses of size s

Input: Dataset (X, y), clause $c \in \mathcal{C}^s(X)$, probabilities p and β;
Output: Classification of c;
Compute $RP_1(c)$ and $RP_0(c)$;
Compute the intervals $\Gamma_1^s(X), \Gamma_0^s(X), \Delta_1^s(X), \Delta_0^s(X)$;
if $RP_1(c) \notin \Gamma_1^s(X)$ **or** $RP_0(c) \notin \Gamma_0^s(X)$ **then**
  Return *Dormant*
else if $RP_1(c) \notin \Delta_1^s(X)$ **or** $RP_0(c) \notin \Delta_0^s(X)$ **then**
  Return *Generating*
else
  Return *Ambiguous*

2.3.2 Computing the RP-statistics for all Clauses of Size 2

Applying the classification procedure to all clauses of size 2 requires computing $RP_1(c)$ and $RP_0(c)$ for every clause $c \in \mathcal{C}^2(X)$. We show that this can be done efficiently through matrix multiplication.

We split the $N \times P$-matrix X of a dataset (X, y) into the $N_1 \times P$-matrix submatrix X_1 with all cases and the $N_0 \times P$-matrix submatrix X_0 with all controls:

$$X_1 = \{X_{i\bullet} : y_i = 1\}$$
$$X_0 = \{X_{i\bullet} : y_i = 0\}.$$

Then, we obtain $RP_1(c)$ for all clauses $c \in \mathcal{C}^2(X)$ by pre-multiplying X_1 by its transpose:

$$M_1 = X_1' X_1.$$

Observe that the i^{th} diagonal entry of the matrix M_1 counts the number of cases where feature i contains a 1, so that $(M_1)_{ii} = RP_1(c)$ for the clause $c = \{i\}$ of size 1. Similarly, for $i \neq j$, off-diagonal entry $(M_1)_{ij}$ is the number of cases where both features i and j have a 1, so that $(M_1)_{ij} = RP_1(c)$ for the clause $c = \{i, j\}$ of size 2. Hence, the values $RP_1(c)$ for all clauses $c \in \mathcal{C}^2(X)$ can be found in the off-diagonal entries of M_1.

Similarly, the off-diagonal entries of the matrix

$$M_0 = X_0' X_0$$

contain the values $RP_0(c)$ for all clauses $c \in \mathcal{C}^2(X)$. Therefore computing $RP_1(c)$ and $RP_0(c)$ for all $c \in \mathcal{C}^2(X)$ has the same complexity as matrix multiplication, namely $\mathcal{O}(P^2 N)$.

Example 3 *[Continuation of Example 2]. In Example 2, we found a generating clause of size 1. We assume this clause has been eliminated, i.e., all samples satisfying the clause* $\{1\}$ *have been removed, so we have a remaining dataset of size* $N = 503, P = 999$.

For the remaining dataset (X, y)*, we compute the domains*

$$\Gamma_1^2(X) = [153, \infty), \quad \Gamma_0^2(X) = [0, 21],$$
$$\Delta_1^2(X) = [0, 98], \quad \Delta_0^2(X) = [90, \infty).$$

Next, for all clauses $c \in \mathcal{C}^2(X)$ of size 2, we compute $RP_1(c)$ and $RP_0(c)$ and apply the classification procedure, see Table 3.

$c \in \mathcal{C}^2(X)$	{2,3}	{2,4}	{2,5}	...	{101,102}	...	{998,1000}	{999,1000}
$RP_1(c)$	60	56	61	...	192	...	72	68
$RP_0(c)$	142	142	135	...	13	...	132	147
$Classify(c)$	D	D	D	...	G	...	D	D

Table 3: RP_1, RP_0 and classifications of a subset of size-2 clauses for the example dataset.

Observe that we classify clause $\{101, 102\}$ as generating, because $RP_1(\{101, 102\}) = 192 \in \Gamma_1^2(X)$, $RP_0(\{101, 102\}) = 13 \in \Gamma_0^2(X)$ and $RP_1(\{101, 102\}) = 192 \notin \Delta_1^2(X)$. All other clauses $c \in \mathcal{C}^2(X), c \neq \{101, 102\}$ have been (correctly) classified as dormant, because $RP_1(c) \notin \Gamma_1^2(X)$ or $RP_0(c) \notin \Gamma_0^2(X)$.

2.3.3 Pools of Promising Clauses

Since it is computationally infeasible to classify all clauses of sizes $s \geq 3$, we develop a procedure that iteratively constructs pools of promising clauses, of which each clause will be classified, with the purpose of encountering generating clauses of size $s \geq 3$. As a starting pool, we consider the set $\mathcal{C}^2(X)$ of all clauses of size 2. For every clause $c \in \mathcal{C}^2(X)$, we compute $RP_1(c)$ and $RP_0(c)$, and use them to compute a fitness value $fit(c)$. For this, we use the fitness function

$$fit(c) = \frac{N_0}{N_1} \cdot RP_1(c) - RP_0(c) \,, \tag{2}$$

where N_1 and N_0 denote the number of cases and controls in the dataset, respectively, and the factor N_0/N_1 makes sure that $RP_1(c)$ and $RP_0(c)$ have equal importance.

The fitness function measures how "promising" a given clause c is by exploiting the insight that for generating clauses, $RP_1(c)$ is higher and $RP_0(c)$ is lower than for dormant clauses. It turns out that this separating effect is already noticeable for subclauses of size 2 of generating clauses of size $s > 2$. The following example illustrates this.

Example 4 *Consider the example dataset in which the generating clauses $\{1\}$ and $\{101, 102\}$ have been eliminated so that the current matrix X has $N = 591$ rows and $P = 997$ columns. The only generating clause left in this dataset is $c^* = \{201, 202, 203, 204\}$ of size 4. First we compute the values $RP_1(c)$, $RP_0(c)$ and $fit(c)$ for subclauses of c^* of size 2, see Table 4. Next, we compute summary statistics for clauses of size 2 that are not a subclause of c^*, i.e., for clauses $c \in \mathcal{C}^2(X) : c \cap c^{\mathcal{G}} = \emptyset$, see Table 5.*
Observe that the subclauses of size 2 of the generating clause c^* can be clearly separated from the other clauses of size 2 by higher fitness values: all subclauses of c^* have $fit(c) \geq 213$, whereas all clauses that are not a subclause of c^* have $fit(c) \leq 136$.

To obtain a pool of promising clauses of degrees 3 and 4, we select the 100 fittest clauses from $\mathcal{C}^2(X)$ and denote them by $\mathcal{F}_{100}(\mathcal{C}^2(X))$. In case of ties, we prioritize clauses with a

$c \in \mathcal{C}^2(x) : c \subseteq c^*$	{201,202}	{201,203}	{201,204}	{202,203}	{202,204}	{203,204}
$RP_1(c)$	43	44	42	41	41	43
$RP_0(c)$	107	111	109	113	110	111
$fit(c)$	235.05	239.0	225.09	213.14	216.14	231.05

Table 4: RP_1, RP_0 and classifications of a subset of size-2 clauses for the example dataset.

	mean	minimum	median	maximum
$RP_1(c)$	16.54	2	16	33
$RP_0(c)$	131.19	85	131	175
$fit(c)$	0.35	-128.18	0.18	135.68

Table 5: Mean, minimum, median and maximum RP_1, RP_0 and $fit(c)$ for dormant clauses of size 2 that are not a subclause of a generating clause of example 4.

lower value $RP_0(c)$. Each pair of these 100 fittest clauses is merged into a new clause of degree 3 or 4. To be precise, the new pool is given by

$$pool_{3,4} = \mathcal{F}_{100}(\mathcal{C}^2(X)) \boxed{\times} \mathcal{F}_{100}(\mathcal{C}^2(X)) \ ,$$

where the operator $\boxed{\times}$ constructs for every pair of nonequal clauses $u = \{u_1, u_2\} \in \mathcal{F}_{100}(\mathcal{C}^2(X))$ and $v = \{v_1, v_2\} \in \mathcal{F}_{100}(\mathcal{C}^2(X))$ with $u \neq v$ the merged clause $u \cup v = \{u_1, u_2, v_1, v_2\}$. As a small example, suppose we consider only the fittest four clauses and that they are given by

$$\mathcal{F}_4(\mathcal{C}^2(X)) = \Big\{\{1,2\},\{1,3\},\{4,5\},\{5,6\}\Big\} \ .$$

Then, the new pool would be given by

$$\mathcal{F}_4(\mathcal{C}^2(X)) \boxed{\times} \mathcal{F}_4(\mathcal{C}^2(X)) = \Big\{\{1,2,3\},\{1,2,4,5\},\{1,2,5,6\},\{1,3,4,5\},\{1,3,5,6\},\{4,5,6\}\Big\} \ .$$

Observe that $pool_{3,4}$ is of size at most $\binom{100}{2}$. In practice, however, the pool is often considerably smaller, since merging elements typically results in a considerable number of double occurences, such as in the following application of the operator $\boxed{\times}$:

$$\Big\{\{1,2\},\{1,3\},\{2,3\}\Big\} \boxed{\times} \Big\{\{1,2\},\{1,3\},\{2,3\}\Big\} = \Big\{\{1,2,3\}\Big\} \ .$$

We classify every clause $c \in pool_{3,4}$ and if we encounter a generating clause, then this completes our procedure for finding a generating clause of size $s \geq 3$ for the current iteration. Example 5 below illustrates how a generating clause of size 4 is found in a small dataset. Alternatively, if we come accross an ambiguous clause, then we report it as a candidate for being generating but for which we cannot exclude confidently that it is not dormant.

If all clauses in $pool_{3,4}$ are classified as dormant, then we repeat the same steps to obtain a pool of clauses of sizes 4 until 8. This new pool is given by

$$pool_{4-8} = \mathcal{F}_{100}(pool_{3,4}) \boxed{\times} \mathcal{F}_{100}(pool_{3,4}) \ .$$

Again, if we classify a clause of $pool_{4-8}$ as generating or ambiguous, then this completes our procedure of finding a generating clause of size $s \geq 3$ for the current iteration. If all clauses in $pool_{4-8}$ are classified as dormant, we repeat the same procedure for the pool

$$pool_{5-16} = \mathcal{F}_{100}(pool_{4-8}) \boxed{\times} \mathcal{F}_{100}(pool_{4-8}) \ .$$

Since we assume that clauses are at most of size 10, we will not create another pool after $pool_{5-16}$. Instead, classifying all clauses in $pool_{5-16}$ as dormant serves as one of the stopping conditions for our algorithm.

Example 5 *For illustration purposes, we consider a small dataset (X, y) with $P = 8$ features and $N = 1,000$ rows. Suppose that the dataset contains one generating clause, namely $c^* = \{1, 2, 3, 4\}$ of size 4. We assume that the procedures of previous sections for finding generating clauses of sizes 1 and 2 have been executed and that all clauses of sizes 1 and 2 were classified as dormant.*

First, we will select the eight fittest clauses of $\mathcal{C}^2(X)$. In the spirit of section 2.3.2, we enumerate the clauses of size 2 as the off-diagonal entries of a matrix that has the set $\mathcal{C}^1(X)$ on its axes. In Table 6, only the off-diagonal entries are filled: for each combination of two features, the corresponding clause $c \in \mathcal{C}^2(X)$ is given, as well as its fitness value. For instance, we find clause $\{1, 3\}$ in row $\{1\}$ and column $\{3\}$, and it has $fit(\{1, 3\}) = 388$. We highlight the eight fittest clauses in blue.

$\mathcal{C}^1(X)$	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
{1}		{1,2} 411	{1,3} 388	{1,4} 371	{1,5} 153	{1,6} 150	{1,7} 182	{1,8} 169
{2}			{2,3} 399	{2,4} 364	{2,5} 175	{2,6} 165	{2,7} 172	{2,8} 125
{3}				{3,4} 387	{3,5} 123	{3,6} 154	{3,7} 159	{3,8} 132
{4}					{4,5} 102	{4,6} 113	{4,7} 169	{4,8} 122
{5}						{5,6} 56	{5,7} -9	{5,8} 16
{6}							{6,7} 77	{6,8} 49
{7}								{7,8} 41
{8}								

Table 6: All clauses $c \in \mathcal{C}^2(X)$, with their fitness value $f(c)$.

In Table 6 we observe that the eight fittest clauses of size 2 are

$$\mathcal{F}_8(\mathcal{C}^2(X)) = \Big\{\{1, 2\}, \{2, 3\}, \{1, 3\}, \{3, 4\}, \{1, 4\}, \{2, 4\}, \{1, 7\}, \{2, 5\}\Big\} \ .$$

Next, we classify each clause in

$$pool_{3,4} = \mathcal{F}_8(\mathcal{C}^2(X)) \boxed{\times} \mathcal{F}_8(\mathcal{C}^2(X)) \ .$$

For this we use the following generating and dormant domains for clauses of size 3 and 4.

s	$\Gamma_1^s(X)$	$\Gamma_0^s(X)$	$\Delta_1^s(X)$	$\Delta_0^s(X)$
3	$[88, \infty)$	$[0, 15]$	$[0, 21]$	$[91, \infty)$
4	$[38, \infty)$	$[0, 10]$	$[0, 13]$	$[40, \infty)$

Table 7: The domains for clauses of size 3 and 4.

We enumerate $pool_{3,4}$ in the following matrix that has the set $\mathcal{F}_8(\mathcal{C}^2(X))$ as its axes. In the following table, the clauses $c \in pool_{3,4}$ can be found in the entries above the diagonal. For each clause we report the triplet $(RP_1(c), RP_0(c), classification(c))$ where the classification yields one of the labels 'G' (generating), 'D' (dormant), or 'A' (ambiguous). We highlight the clauses labelled as generating in green. One can verify the classifications with the intervals given above. We classify every clause, independent of whether it is of size 3 or 4. If we find multiple generating clauses, we will select the fittest one.

$\mathcal{F}_8(\mathcal{C}^2(X))$	{1,2}	{2,3}	{1,3}	{3,4}	{1,4}	{2,4}	{1,7}	{2,5}
{1,2}		{1,2,3} (63,64,D)	{1,2,3} (63,64,D)	{1,2,3,4} (57,1,G)	{1,2,4} (63,79,D)	{1,2,4} (63,79,D)	{1,2,7} (40,88,D)	{1,2,5} (41,92,D)
{2,3}			{1,2,3} (63,64,D)	{2,3,4} (61,54,D)	{1,2,3,4} (57,1,G)	{2,3,4} (61,54,D)	{1,2,3,7} (36,31,D)	{2,3,5} (37,86,D)
{1,3}				{1,3,4} (62,63,D)	{1,3,4} (62,63,D)	{1,2,3,4} (57,1,G)	{1,3,7} (39,83,D)	{1,2,3,5} (34,34,D)
{3,4}					{1,3,4} (62,63,D)	{2,3,4} (61,54,D)	{1,3,4,7} (36,29,D)	{2,3,4,5} (32,26,D)
{1,4}						{1,2,4} (63,79,D)	{1,4,7} (41,91,D)	{1,2,4,5} (36,40,D)
{2,4}							{1,2,4,7} (37,35,D)	{2,4,5} (38,101,D)
{1,7}								{1,2,5,7} (21,45,D)
{2,5}								

Table 8: All clauses $c \in pool_{3,4}$, with the tuple $(RP_1(c), RP_0(c), classify(c))$.

From Table 8, we conclude that clause $\{1, 2, 3, 4\}$ is generating and that all other clauses in this pool are dormant.

2.4 Elimination

Previously, when searching for a generating clause of size s, we have consistently assumed that the dataset (X, y) did not contain clauses of smaller sizes $1, ..., (s-1)$. The presence of a generating clause of smaller size $1, ..., (s-1)$ may hamper the likelihood of finding a generating clause of size s. Therefore, as soon as we classify a clause as generating, we eliminate it from the dataset.

When we eliminate a clause $c = \{c_1, ..., c_s\}$, we remove the rows of matrix X and vector y where clause c is active. Moreover, we remove the columns $c_1, ..., c_s$ from the matrix X.

The obtained dataset (X', y') is given by

$$X' = \Big\{ X_{ij} : \bigwedge_{j \in c} X_{ij} \neq 1 \text{ and } j \notin c \Big\}$$
$$y' = \Big\{ y_i : \bigwedge_{j \in c} X_{ij} \neq 1 \Big\} .$$

The elimination procedure is illustrated with a small example.

Example 6 *Consider the following small dataset (X, y) with $N = 5$ rows and $P = 6$ columns. Suppose that the clause $c = \{2, 5\}$ is classified as generating and eliminated. Observe that columns 2 and 5 as well as rows 1 and 4, where $\{2, 5\}$ is active, are removed.*

$$X = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}, \; y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \xrightarrow{\textit{Eliminate } \{2,5\}} X' = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 3 & 4 & 6 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array}, \; y' = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Eliminating a generating clause leads to generating clauses of bigger sizes being better separated from dormant clauses. Some intuition for this can be obtained by reconsidering Table 1 and to aim attention at the columns of the generating clause $\{201, 202, 203, 204\}$ of size 4. Most of the samples for which $y_i = 1$ are cases because either clause $\{1\}$ or clause $\{101, 102\}$ is active, while $\{201, 202, 203, 204\}$ is not. As a result, the columns of features 201 up to 204 resemble columns belonging to dormant clauses. Hence, the presence of uneliminated cases related to clauses of small size hampers the overall distinguishability of generating and dormant clauses of greater size, and thus the functioning of our algorithm.

To measure how "well-separated" generating and dormant domains are, we introduce the *Degree of Overlap* (DoO) statistics. For a dataset with N_1 cases, we let $DoO_1^s(N, N_1)$ be the probability that, for a generating clause $c \in \mathcal{G}^s(X)$, the value $RP_1(c)$ lies in the dormant domain, i.e.,

$$\begin{aligned} DoO_1^s(N, N_1) &= \mathbb{P}\big[RP_1(c) \in \Delta_1^s(X) \;\big|\; c \in \mathcal{G}^s(X)\big] \\ &= G_1^s((\delta_1^s)^{-1}(1)) \, . \end{aligned}$$

Similarly, for a dataset with N_0 controls, we let $DoO_0^s(N, N_0)$ be the probability that $RP_0(c)$ of a generating clause $c \in \mathcal{G}^s(X)$ lies in the dormant domain, i.e.,

$$\begin{aligned} DoO_0^s(N, N_0) &= \mathbb{P}\big[RP_0(c) \in \Delta_0^s(X) \;\big|\; c \in \mathcal{G}^s(X)\big] \\ &= 1 - G_0^s((\delta_0^s)^{-1}(1)) \, . \end{aligned}$$

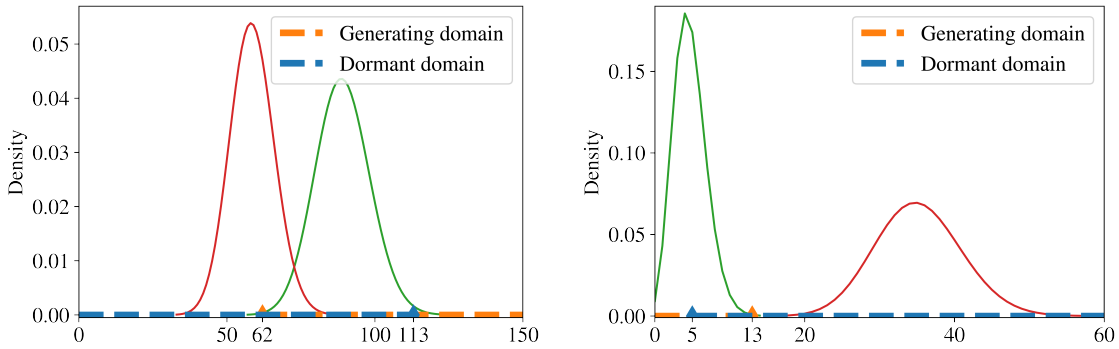
The lower the degrees of overlap are, the better the dormant and generating domains are separeted. In fact, recalling the definitions of the generating domain $\Gamma_1^s(X)$ and $\Gamma_0^s(X)$, it is easily verified that

$$\Gamma_1^s(X) \cap \Delta_1^s(X) = \emptyset \iff DoO_1^s(N, N_1) < 0.001,$$
$$\Gamma_0^s(X) \cap \Delta_0^s(X) = \emptyset \iff DoO_0^s(N, N_0) < 0.001.$$

The crucial insight for justifying elimination is that elimination of a generating clause $c \in \mathcal{G}^s(X)$ of size s reduces the degrees of overlap for any generating clause $c' \in \mathcal{G}^{s'}(X)$ of bigger size $s' > s$. To see how $DoO_1^{s'}(N, N_1)$ diminishes, recall that the distribution centers of RP_1 for clauses of size s' are given by $N \cdot p^{s'} \cdot (1-\beta)$ for generating clauses and by $N_1 \cdot p^{s'}$ for dormant clauses. The rows where clause c' is active are predominantly cases. Hence, eliminating c will reduce N_1 by a considerably larger portion than N. Consequently, the dormant domain $\Delta_1^{s'}(X)$ shifts further to the left than the generating domain $\Gamma_1^{s'}(X)$.

Similarly, recall that the distribution centers of RP_0 for clauses of size s' are $N \cdot p^{s'} \cdot \beta$ for generating clauses and $N_0 \cdot p^{s'}$ for dormant clauses. Since eliminating c will reduce N more than N_0, the degree of overlap $DoO_0^{s'}(N, N_0)$ will decrease, enabling better distinction between dormant and generating clauses. The following example displays the RP-distributions and the corresponding degrees of overlap for clauses of size 4, before and after elimination of generating clauses of sizes 1 and 2.

Example 7 *Reconsider the example dataset. Before eliminating the generating clauses* $\{1\}$ *and* $\{101, 102\}$*, the degrees of overlap are high for the generating clause* $\{201, 202, 203, 204\}$*. Hence, if* $\{201, 202, 203, 204\}$ *would be classified based on the complete dataset, it is likely the that it is labelled as 'Ambiguous'. In Figure 3, we see that the generating and dormant domains of clauses of size 4 are largely intertwined so that the degrees of overlap are high.*

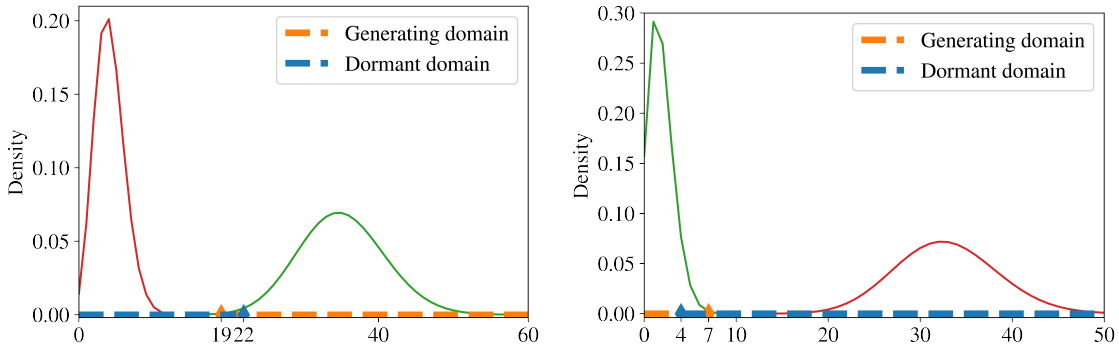


(a) RP_1, with $DoO_1^4(X) = 0.995$.

(b) RP_0, with $DoO_0^4(X) = 0.503$.

Figure 3: Before elimination of clauses $\{1\}$ and $\{101, 102\}$: the RP-distributions of clauses of size 4.

Next, we eliminate the generating clauses $\{1\}$ *and* $\{101, 102\}$*. As a consequence, we see in Figure 4 that the generating and dormant domains for clauses are now better separated and the degrees of overlap have been significantly reduced.*



(a) RP_1, with $DoO_1^4(X) = 0.010$.

(b) RP_0, with $DoO_0^4(X) = 0.116$.

Figure 4: After elimination of clauses $\{1\}$ and $\{101, 102\}$: the RP-distributions of clauses of size 4.

2.5 The Remaining Positives Algorithm (RPA)

In this section we integrate the techniques of the previous sections into a solution algorithm, which we call the *Remaining Positives Algorithm* (RPA). A reader familiar with the techniques of previous subsections could choose to ignore the clarifying text and skip directly to Algorithm 3.

RPA starts by declaring the sets $\mathcal{G}$ and $\mathcal{A}$ that will store the clauses that we classify as generating and ambiguous. They will be the output of the algorithm. The clauses in $\mathcal{G}$ are confidently considered to be generating. The clauses in $\mathcal{A}$ are cautiously considered promising candidates for being generating, but for which further examination is necessary to reach a solid conclusion.

First, clauses of size 1 are found exhaustively by running a subroutine with the set $\mathcal{C}^1(X)$ of all clauses of size 1 as input. The subroutine, on input of a certain pool of clauses, classifies all clauses and elects the winner of the pool, or concludes that the current pool has no winner. If any clauses are labelled as generating, the one with the highest fitness value among them is chosen as the winner, where we recall that the fitness of a clause is computed using equation (2). Otherwise, if any clauses are labelled as ambiguous, the winner of the pool is the fittest among the ambiguous clauses. If a winner is found, it is added to the corresponding set $\mathcal{G}$ or $\mathcal{A}$ and eliminated from the dataset. The subroutine is applied repeatedly to $\mathcal{C}^1(X)$, until no more clauses are classified as generating or ambiguous, which we interpret as that the dataset contains no more generating clauses of size 1.

Observe that, throughout the paper, several concepts have been defined as dependent on the matrix X, such as the sets $\mathcal{C}^s(X)$, $\mathcal{G}^s(X)$ and $\mathcal{D}^s(X)$, the intervals $\Gamma_1^s(X), \Gamma_0^s(X), \Delta_1^s(X)$ and $\Delta_0^s(X)$. In fact, this is a slight misrepresentation, because these objects also depend on the vector y, but for conciseness of notation y is left out. Additionally, we now denote the number of cases $N_1(X)$ as dependent on the matrix X. When X and y change due to elimination, these objects change accordingly.

To find generating clauses of size 1, one could argue that it is more efficient to apply the subroutine to $\mathcal{C}^1(X)$ only once and immediately store all clauses that are classified as

generating or ambiguous in $\mathcal{G}$ or $\mathcal{A}$, respectively. However, throughout the algorithm, we decide to select the single fittest generating or ambiguous clause and eliminate it immediately, before looking for other candidate clauses. We argue that the elimination step 'cleans up' the dataset, and therefore choose to perform elimination every time after the subroutine has found a winner. Since our algorithm is efficient, we can afford to perform elimination more often than, in some cases, may be strictly necessary.

Having completed the procedure for finding clauses of size 1, we repeat the same procedure for clauses of size 2.

To find generating clauses of size greater than or equal to 3, we run the subroutine with as input a carefully constructed succession of pools of clauses. As an initial pool, we take the pool of promising clauses of size 3 and 4, which, we recall from Section 2.3.3, is given by

$$pool_{3,4} = \mathcal{F}_{100}(\mathcal{C}^2(X)) \boxed{\times} \mathcal{F}_{100}(\mathcal{C}^2(X)) \ .$$

If the outcome of running the subroutine with $pool_{3,4}$ is that this pool has a winner clause, then it is eliminated and we reinitialize the procedure of finding clauses of size at least 3. If $pool_{3,4}$ turns out not to have a winner, we run the subroutine with the next pool

$$pool_{4-8} = \mathcal{F}_{100}(pool_{3,4}) \boxed{\times} \mathcal{F}_{100}(pool_{3,4}) \ .$$

If $pool_{4-8}$ has a winner, it is eliminated and we restart the procedure of finding generating clauses of size at least 3 with a new $pool_{3,4}$. If $pool_{4-8}$ does not have a winner, we run the subroutine with the next pool

$$pool_{5-16} = \mathcal{F}_{100}(pool_{5-16}) \boxed{\times} \mathcal{F}_{100}(pool_{5-16}) \ .$$

Similarly, if $pool_{5-16}$ has a winner, it is processed and we restart the process with a new $pool_{3,4}$. Otherwise, if $pool_{5-16}$ does not have a winner, then the algorithm terminates.

RPA boils down to applying the subroutine to a carefully constructed series of pools. We impose three stopping conditions. The 'cleanest' way in which RPA can terminate is when three consecutive subroutines have not elected a winner. In such cases, all clauses in the series of $pool_{3,4}$, $pool_{4-8}$ and $pool_{5-16}$ have been classified as dormant, so that we can quite safely conclude that the dataset contains no more generating clauses. However, it can occur that the algorithm keeps finding ambiguous clauses. Sometimes, RPA manages to explain all cases ($N_1(X) = 0$), which we impose as another stopping condition. Alternatively, if unrestricted, it can occur that the algorithm keeps on finding ambiguous clauses, which tend to be of larger sizes. We consider such 'tails of ambiguous clauses' degenerate behaviour and impose that the algorithm stops if it has found 10 consecutive ambiguous clauses.

By these stopping conditions, and with the assumption that every dataset has a limited number of generating clauses, RPA terminates after a finite number of subroutines. The exact number of executed subroutines depends on the dataset. Assuming that any dataset contains no more than 10 generating clauses, and with the stopping condition after 10 ambiguous clauses, the number of executed subroutines cannot exceed 60. For a given subroutine, the classification of each clause c requires the computation of $RP_1(c)$ and $RP_0(c)$, which is done in time $\mathcal{O}(N)$. The largest pool that we can encounter is $\mathcal{C}^2(X)$, which is of size $\mathcal{O}(P^2)$. Therefore, RPA runs in time $\mathcal{O}(P^2N)$.

Algorithm 3: The Remaining Positives Algorithm (RPA)

Input: Dataset (X, y), probability estimate $\hat{p}$, error estimate $\hat{\beta}$;
Output: Set of found generating clauses $\mathcal{G}$ and ambiguous clauses $\mathcal{A}$;
$\mathcal{G} := \emptyset$, $\mathcal{A} := \emptyset$;
`// Find generating clauses of size 1 and 2 exhaustively`
while Subroutine($\mathcal{C}_1(X)$) finds a winner **do**
  Subroutine($\mathcal{C}_1(X)$);
while Subroutine($\mathcal{C}_2(X)$) finds a winner **do**
  Subroutine($\mathcal{C}_2(X)$);
`// Find generating clauses of size` $s \geq 3$ `using the pooling procedure`
$pool := \mathcal{F}_{100}(\mathcal{C}_2(X)) \boxed{\times} \mathcal{F}_{100}(\mathcal{C}_2(X))$;
$winnerless\_counter := 0$, $ambiguous\_counter := 0$;
while $winnnerless\_counter \leq 3$ and $ambiguous\_counter \leq 10$ and $N_1(y) > 0$ **do**
  Subroutine($pool$);
  if the subroutine did not find a winner **then**
    $pool := \mathcal{F}_{100}(pool) \boxed{\times} \mathcal{F}_{100}(pool)$;
    $winnerless\_counter := winnerless\_counter + 1$;
  else
    `// Reinitialize the pooling procedure`
    $pool := \mathcal{F}_{100}(\mathcal{C}_2(X)) \boxed{\times} \mathcal{F}_{100}(\mathcal{C}_2(X))$;
    $winnerless\_counter := 0$;
Return $\mathcal{G}$ and $\mathcal{A}$.

Subroutine(*pool*):
$\mathcal{G}_{tmp} := \emptyset$, $\mathcal{A}_{tmp} := \emptyset$;
forall Clauses $c \in pool$ **do**
  if $Classify(c) = Generating$ **then**
    $\mathcal{G}_{tmp} := \mathcal{G} \cup \{c\}$;
  else if $Classify(c) = Ambiguous$ **then**
    $\mathcal{A}_{tmp} := \mathcal{A} \cup \{c\}$;
if $\mathcal{G}_{tmp} \neq \emptyset$ **then**
  Select winner $c^* = \arg\max_{c \in \mathcal{G}_{tmp}} fit(c)$ from $\mathcal{G}_{tmp}$;
  $\mathcal{G} := \mathcal{G} \cup \{c^*\}$;
  $(X, y) := Eliminate(X, y, c^*)$;
  $ambiguous\_counter := 0$;
else if $\mathcal{A}_{tmp} \neq \emptyset$ **then**
  Select the winner $c^* = \arg\max_{c \in \mathcal{A}_{tmp}} fit(c)$ from $\mathcal{A}_{tmp}$;
  $\mathcal{A} := \mathcal{A} \cup \{c^*\}$;
  $(X, y) := Eliminate(X, y, c^*)$;
  $ambiguous\_counter := ambiguous\_counter + 1$;
else
  `// This execution of the subroutine did not find a winner`

3. Experiments

Before going into elaborate experiments, we first evaluate how the number of samples N, the number of features P and the feature frequency p affect the performance of the Remaining Positives Algorithm (RPA). For this we use the DoO as well as small experiments.

3.1 The effect of dataset characteristics on the performance of RPA

In Section 2.4, we defined the degree of overlap statistics to justify the elimination procedure. More generally, we can use these statistics as predictors for the quality of the algorithm's output. In particular, for unseen datasets, DoO_1^s and DoO_0^s can help us evaluate the likelihood that RPA will find the generating clauses. The advantage of the statistics DoO_1^s and DoO_0^s is that they are easily computed for clauses of any size s with only a few high-level parameters of a dataset: the frequency p of occurences of a feature, the error probability β, the dimensions $N \times P$ of matrix X, and the number of cases N_1.

However, correctly evaluating DoO_1^s and DoO_0^s is delicate. The exact relationship between the degrees of overlap and the algorithm performance is not characterized in this paper. Here we claim that generating clauses are to be found more likely if at least one of the degrees of overlap is low. In this section, we use these statistics to point out several determinants of RPA's performance and support the claims with experiments.

3.1.1 A Crucial Assumption for Using Degrees of Overlap

Using DoO_1^s and DoO_0^s to evaluate the likelihood of finding a generating clause of size s only makes sense under the assumption that the dataset contains no generating clauses of smaller sizes $1, ..., (s-1)$. By design, RPA finds generating clauses in order of increasing size. If finding a generating clause $c \in \mathcal{G}^s(X)$ of size s is preceded by finding a generating clause of a smaller size, then eliminating the smaller clause reduces the number of rows of the dataset, which affects the likelihood of finding clause c.

To illustrate this, we consider two datasets with dimensions $N = 5,000$ and $P = 1,000$. The first dataset (X_1, y_1) contains a rule with only one generating clause of size 7, i.e.,

$$rule_1 = \big\{\{1,2,3,4,5,6,7\}\big\} . \tag{3}$$

The other dataset (X_2, y_2) additionally contains a generating clause of size 1, i.e.,

$$rule_2 = \big\{\{1,2,3,4,5,6,7\},\{10\}\big\} .$$

To evaluate the likelihood of finding the clause of size 7 in dataset (X_1, y_1), we can directly compute the degrees of overlap with the initial dimensions $(N, P) = (5,000; 1,000)$, because there are no generating clauses of smaller size. In Table 9, we see that at least one of the degrees of overlap is low, i.e., $DoO_1^7(X_1) = 0.01$, so that we may expect the clause to be found with high likelihood. On the other hand, for the dataset (X_2, y_2), RPA will search for the clause of size 7 only after the clause of size 1 is eliminated. We denote the dataset resulting from eliminating the clause $\{10\}$ by (X_2^E, y_2^E). Since eliminating the clause of size 1 will expectedly halve the number of rows, we have to compute the degrees of overlap with $N^E = 2,500$ to obtain a realistic prognosis of the likelihood of finding the clause of size 7.

In Table 9, we see that the degrees of overlap are high, so we can expect it to be less likely that the clause of size 7 is found by RPA.

Indeed, in the following experiment, we find that the generating clause of size 7 is found more often in datasets with $rule_1$ than in datasets with rule $rule_2$. For both rules, we perform our algorithm on 100 generated datasets with dimensions $N = 5,000$ and $P = 1,000$, and $p = 0.5$ and $\beta = 0.05$. In Table 9, we see that the generating clause of size 7 is found considerably more often in datasets with $rule_1$.

	(N, P)	DoO_1^7	DoO_0^7	Fraction of generating clauses of size 7 retrieved
$rule_1$	(5,000;1,000)	0.01	1.00	0.44
$rule_2$	(2,500;999)	0.60	1.00	0.05

Table 9: For two settings, we compute the degrees of overlap for clauses of size 7 with the dimensions (N, P) at the appropriate stage of RPA: i.e., after eliminating generating clauses of smaller size. For 100 datasets generated for both settings with $p = 0.5$, we apply the algorithm and report the fraction of found generating clauses of size 7.

Henceforth, in order to be able to interpret the degrees of overlap correctly, we assume that every dataset contains only one generating clause.

3.1.2 Effect of dimensions N and P

Generating clauses are found significantly more easily if the number of rows N is high, but the effect of the number of columns P is negligible. Recall that N, N_1 and N_0 play an important role in the location of the RP-distributions. It can be verified that larger N lead to a better separation between generating and dormant clauses.

The number of columns P affects the number of dormant clauses. At most a few clauses are generating, so that the number of dormant clauses of size s is approximately $|\mathcal{C}^s(X)| = \binom{P}{s}$. By construction, the dormant domains are wider for larger P, so that $RP_1(c)$ and $RP_0(c)$ of any dormant clause $c \in \mathcal{D}^s(X)$ materialize within the dormant domains with probability almost 1. This could cause the degree of overlap to increase for higher P. However, if N is large enough, the generating and dormant domains are sufficienlty far apart that a large P hardly affects the degrees of overlap.

We reconsider $rule_1$ given in equation (3). In Table 10 we report the degrees of overlap for clauses of size 7, for different dataset sizes. The degrees of overlap suggest that the generating clauses of size 7 are unlikely to be found for smaller datasets with $N = 1,000$ rows, but are easily found for large datasets with $N = 10,000$ rows. The number of columns P makes an insignificant difference.

The following experiment confirms our prediction of the likelihood of finding the generating clause of size 7. For different dimensions N and P of the matrix X, we generated 100 datasets with $p = 0.5$ and $\beta = 0.05$. In Table 10 we report the fraction of generating clauses of size 7 retrieved by RPA.

(N, P)	DoO_1^7	DoO_0^7	Fraction of generating clauses of size 7 retrieved
(1,000;1,000)	0.99	1.00	0.00
(1,000; 2,500)	0.99	1.00	0.00
(10,000;1,000)	0.00	0.00	0.97
(10,000;2,500)	0.00	0.01	0.87

Table 10: For four settings with $rpf = [7]$, with $p = 0.5$ and different dimensions (N, P), we compute the degrees of overlap for clauses of size 7. For each setting, we generate 100 datasets and apply RPA. We report the fraction of found generating clauses.

3.1.3 Effect of Clause Size and Probability p

The larger a generating clause is, the harder it is to find. Also, the smaller the probability p is, the harder it is to find generating clauses of any size. To explain both effects, we recall that the probability p^s that a clause of size s is active plays a major role in the derivation of the RP-distributions. If p^s is low, there will be few rows where generating clauses of size s are active, and the problem of discerning them with any method becomes harder. The smaller p^s is, the higher are the degrees of overlap.

Table 11 reports the degrees of overlap for various p and clause sizes, for datasets with dimensions $N = 5,000$ and $P = 1,000$ and with $\beta = 0.05$. We observe that for $p = 0.5$ and for all reported clause sizes, at least one of the degrees of overlap is low, so that generating clauses will be likely found by RPA. For lower values of p, the degrees of overlap increase. For $p = 0.125$, generating clauses of sizes 1 and 3 are likely to be found, but generating clauses of sizes 5 and 7 seem to be indistinguishable from dormant clauses.

Our expectations regarding the likelihood of finding generating clauses are confirmed in the following experiment. For every reported combination of probability p and clause size s, we generate 100 datasets with $\beta = 0.05$ that contain one generating clause of size s. Table 11 reports for each parameter setting the fraction of generating clauses that was retrieved by RPA. Now, we distinguish generating clauses that the algorithm found by labelling them as generating ('G') or ambiguous ('A'). In Table 11, we see that for $p = 0.125$, generating clauses of size 3 are only moderately distinguishable from dormant clauses. Table 11 confirms this, since a considerable fraction of the generating clauses in this context were not classified as generating but as ambiguous.

3.2 Setup of further experiments

We test the performance of RPA in various settings. For each setting, we generated 100 datasets and ran the algorithm on each of them. Section 3.1 demonstrated that the performance depends on the boolean rule, the dimensions (N, P) of the dataset, and the frequency p of occurences of a feature. We construct 16 settings that vary in these aspects. Throughout, we use the constant error probability $\beta = 0.05$.

We run experiments for 8 settings with probability $p = 0.5$, which are reported in Section 3.3, and 8 settings with probability $p = 0.25$, which are reported in Section 3.4. For each p, we test for two boolean rules. Henceforth, we will represent boolean rules by their *rule profiles* (rpf), which is an enumeration of the sizes of the clauses that the rule contains. We

p	s	DoO_1^s	DoO_0^s	Fraction of generating clauses retrieved	
				Labelled 'G'	Labelled 'A'
0.5	1	0.00	0.00	1.00	0.00
	3	0.00	0.00	1.00	0.00
	5	0.00	0.00	1.00	0.00
	7	0.01	1.00	0.55	0.00
0.25	1	0.00	0.00	0.99	0.00
	3	0.00	0.00	0.99	0.00
	5	0.98	1.00	0.00	0.00
	7	1.00	1.00	0.00	0.00
0.125	1	0.00	0.00	1.00	0.00
	3	0.42	1.00	0.54	0.08
	5	1.00	1.00	0.00	0.00
	7	1.00	1.00	0.00	0.00

Table 11: For twelve settings with different $p \in \{0.5, 0.25, 0.125\}$, with $(N, P) = (5,000; 1,000)$ and with rule profile $rpf = [s]$ with one generating clause of size s for $s \in \{1, 3, 5, 7\}$, we compute the degrees of overlap for clauses of size s. For each setting, we generated 100 datasets, applied RPA. This table reports the fraction of generating clauses retrieved, where we seperate clauses that the algorithm labelled as 'G' (generating) and 'A' (ambiguous).

test boolean rules with the profiles

$$rpf_1 = [1, 2, 3, 5],$$
$$rpf_2 = [6, 7, 8].$$

Here, rpf_1 describes a boolean rule with one generating clause of size 1, one of size 2, one of size 3, and one of size 5. The first rule profile tests RPA's success on small clauses, which can be found relatively easily. The second rule tests for larger clauses, which are found less easily. For each rule profile, we generate datasets of four different sizes, namely for $N = 1,000$ and $N = 10,000$ rows, and for $P = 1,000$ and $P = 2,500$ columns. To sum up, we test for the dataset sizes

$$(N, P) \in \Big\{(1,000; 1,000),\ (1,000; 2,500),\ (10,000; 1,000),\ (10,000; 2,500)\Big\}.$$

Recall that RPA has the objective of uncovering the generating clauses that produced the data and that it gives as output a collection of clauses where each is labelled either as 'G', if we concluded that it is almost certainly generating, or 'A', if it is a promising but ambiguous candidate. To evaluate the performance of the algorithm, we will not only report standard performance measures such as accuracy and runtime, because they may paint a deceiving picture. For instance, on input of a dataset with a small fraction of cases, the algorithm could achieve a high accuracy by simply returning an empty rule, which would clearly be undesirable. Instead, our focus will be on the extent to which RPA succeeds in finding the generating clauses.

3.3 Results for $p = 0.5$

We test RPA for 8 settings where datasets are generated with probability $p = 0.5$. First, we observe how often the generating clauses of the rule are found, where we separate the results by clause size: see Table 12.

The fourth and fifth column show the fraction of the generating clauses that were found, where we distinguish clauses that our algorithm labelled as generating ('G') or ambiguous ('A'). Consistently, we see that smaller clauses are found more easily. Moreover, a larger number of rows (N) increases the likelihood of finding generating clauses, but the number of columns (P) makes little difference. There seems to be a natural barrier, depending on the precise setting, that determines with what likelihood generating clauses are found. This is most apparent in the results for the rule profile $[6, 7, 8]$. In settings with $N = 1,000$ rows, the generating clauses of sizes 6, 7 and 8 are hardly found. With $N = 10,000$ rows, however, the generating clauses of sizes 6 and 7 are found with high likelihood, but those of size 8 are still practically unfindable. Such barriers depend on several characteristics of the dataset and can be explained by the degrees of overlap (cf. Section 3.1).

The last two columns of Table 12 reveal the nature of the clauses that were labelled as 'G' or 'A'. In almost all settings and for almost all sizes, the event that a clause receives the lable 'G' is an almost certain guarantee for it to be generating. Exceptions are those cases where generating clauses are found with low but nonnegligible probability: such as in $rpf = [1, 2, 3, 5]$, $(N, P) = (1,000; 1,000)$, $size = 5$ and $rpf = [6, 7, 8]$, $(N, P) = (1,000; 1,000)$, $size = 6$. Such borderline cases can be recognized by computing the degrees of overlap for the respective circumstances. In such cases, we know that we should be more cautious about concluding that clauses labelled as 'G' are generating. As for the clauses labelled 'A', we see that they are sometimes generating, but mostly they are not.

In Table 13, we further explore the clauses labelled as 'A'. In almost all observed executions of RPA, the clauses labelled 'G' are found before the clauses labelled 'A', as can be seen in the third column of Table 13. After a number of clauses have been labelled as 'G' and the corresponding samples have been eliminated, a number of unexplained cases could remain for two possible reasons. Possibly, there is another generating clause that has not been found yet. Alternatively, recall that a number of cases are due to error instead of a generating clause. Attempting to explain these remaining cases, the algorithm may produce a 'tail' of clauses labelled 'A'.

Most clauses labelled 'A' are not generating, but some contain features that are part of a generating clause. The last four columns of Table 13 show the fraction of features that belong to a generating clause. For some clauses labelled 'A', all features belong to a generating clause. This could occur when a subclause of size 4 of a generating clause of size 5 is prematurely labelled 'A' in the pool with promising clauses of sizes 3 and 4 ($pool_{3,4}$). Consequently, this generating clause of size 5 never reaches the pool with clauses of sizes between 4 and 8 ($pool_{4-8}$), where it could have correctly been labelled as 'G'. Some clauses labelled 'A' contain a number of features belonging to a generating clause, but most contain no features of interest.

Table 14 reports the average accuracy, sensitivity and specificity on the training data resulting from the clauses obtained for the 100 datasets that were tested. For each dataset, we predicted the dependent vector y with two sets of clauses: first only with the clauses

rpf	(N, P)	Clause size	Of all generating clauses		Of all clauses labelled 'G'	Of all clauses labelled 'A'
			% Label 'G'	% Label 'A'	% Generating	% Generating
[1,2,3,5]	(1,000; 1,000)	1	1.00	0.00	1.00	N/A
		2	1.00	0.00	1.00	N/A
		3	0.99	0.00	1.00	N/A
		5	0.02	0.02	0.50	0.14
	(1,000; 2,500)	1	1.00	0.00	1.00	N/A
		2	0.99	0.00	1.00	N/A
		3	1.00	0.00	1.00	N/A
		5	0.00	0.00	0.00	0.00
	(10,000; 1,000)	1	1.00	0.00	1.00	N/A
		2	1.00	0.00	1.00	N/A
		3	1.00	0.00	1.00	N/A
		5	1.00	0.00	1.00	N/A
	(10,000; 2,500)	1	0.99	0.00	1.00	N/A
		2	1.00	0.00	1.00	N/A
		3	1.00	0.00	1.00	N/A
		5	1.00	0.00	1.00	N/A
[6,7,8]	(1,000; 1,000)	6	0.15	0.03	0.84	0.02
		7	0.00	0.00	0.00	0.00
		8	0.00	0.00	N/A	0.00
	(1,000; 2,500)	6	0.03	0.01	0.83	0.03
		7	0.01	0.00	1.00	0.00
		8	0.00	0.00	0.00	0.00
	(10,000; 1,000)	6	1.00	0.00	1.00	N/A
		7	0.94	0.00	1.00	N/A
		8	0.07	0.00	1.00	N/A
	(10,000; 2,500)	6	1.00	0.00	1.00	N/A
		7	0.89	0.00	1.00	N/A
		8	0.02	0.00	1.00	N/A

Table 12: For eight settings with $p = 0.5$, rule profile $[1, 2, 3, 5]$ or $[6, 7, 8]$ and with different dimensions (N, P), RPA has been applied to 100 generated datasets. The fraction of retrieved generating clauses is reported here.

labelled as 'G' (the middle three columns of Table 14), and then with all clauses found by our algorithm (the last three columns of Table 14). Generally, the accuracy is appropriately around $(1 - \beta) = 0.95$. The sensitivity and specificity metrics give us more insight into the behaviour of RPA. When we include the clauses labelled 'A' in the prediction, we see the sensitivity increase to inappropriate heights, given the error rate $\beta = 0.05$. Hence, the clauses labelled 'A' cause an overfitting to the cases. Predictions are best made with only the clauses labelled 'G', and the clauses labelled 'A' should be considered as a set of promising features that could guide further research.

For a proper interpretation of the values in Table 14, one should know the composition of the cases in the datasets. Of the observed cases, we call those caused by the boolean rule *true cases* and those caused by error *error cases*. One should only aim to predict the true cases, so knowing the relative proportions guides us to a correct interpretation of the sensitivity values. In the table, we include columns for the expected fraction of true cases and error cases for the two rule profiles. To obtain the fraction of true cases, we note that

rpf	(N, P)	Fraction of runs with labels 'G' before labels 'A'	Clauses labelled 'A' that are not generating					
			Number per dataset	Size	% Features contained in a generating clause			
			mean	mean	mean	min	med	max
[1,2,3,5]	(1,000; 1,000)	0.97	3.4	5.4	0.08	0.00	0.00	1.00
	(1,000; 2,500)	1.00	3.3	5.1	0.05	0.00	0.00	1.00
	(10,000; 1,000)	1.00	3.9	8.0	0.00	0.00	0.00	0.00
	(10,000; 2,500)	1.00	5.8	8.0	0.00	0.00	0.00	0.00
[6,7,8]	(1,000; 1,000)	0.99	7.6	7.0	0.07	0.00	0.00	1.00
	(1,000; 2,500)	0.99	6.8	6.9	0.04	0.00	0.00	0.83
	(10,000; 1,000)	1.00	0.0	9.6	0.63	0.56	0.60	0.78
	(10,000; 2,500)	1.00	0.0	9.6	0.46	0.00	0.56	0.88

Table 13: For eight settings with $p = 0.5$, rule profile $[1, 2, 3, 5]$ or $[6, 7, 8]$ and with different dimensions (N, P), RPA has been applied to 100 generated datasets. This table provides insight in the quality of the clauses labelled 'A'.

a row $i \in [N]$ is a true case if at least one generating clause is active and the error is not, which occurs with probability

$$\mathbb{P}[y_i = 1 \wedge \epsilon_i = 0] = \Big(1 - \prod_{s \in rpf} (1 - p^s)\Big) \cdot (1 - \beta) \; . \tag{4}$$

To obtain the fraction of error cases, we note that a row $i \in [N]$ is an error case if all generating clauses are inactive and the error is active, which occurs with probability

$$\mathbb{P}[y_i = 0 \wedge \epsilon_i = 1] = \prod_{s \in rpf} (1 - p^s) \cdot \beta \; . \tag{5}$$

The third and fourth column of Table 14 provide the expected fraction of samples that are true cases and error cases, respectively, where these values have been computed using equations (4) and (5). Observe that for the rule profile $rpf_1 = [1, 2, 3, 5]$, the vast majority of cases are true cases: specifically, $0.64/(0.64 + 0.02) = 0.97$. Hence, it is reasonable that high sensitivities are achieved. However, for the rule profile $rpf_2 = [6, 7, 8]$, only a fraction 0.03 of samples are true cases and 0.05 are error cases. Therefore, any sensitivity greater than $0.03/(0.03 + 0.05) = 0.38$ should cause one to suspect overfitting and, similarly, the observed sensitivity values 0.30 and 0.29 for the datasets in this setting with $N = 10,000$ rows are appropriate. Note that here we report accuracy, sensitivity and specificity of the training dataset, as the aim is to show the effect of certain data assumptions on the number of true versus error cases. The assessment of the generalizability lies in evaluating whether RPA retrieves the true underlying AND clauses, rather than in accuracy, sensitivity and specificity of a test dataset.

Finally, Table 16 displays the runtimes for each of the settings. We report the total runtime, but also decompose it into the time needed to find the clauses labelled 'G' and 'A'. Observe that the smaller instances are solved within fifteen minutes, and that all instances except a few with dimensions $(N, P) = (10,000; 2,500)$ are solved within an hour. The time needed to find the clauses labelled 'G' is rather stable for each setting. However, in some settings, the runtime needed to find the clauses labelled 'A' shows much more variation.

rpf	(N, P)	% Samples that are true cases	% Samples that are error cases	Using only clauses labelled 'G'			Using all found clauses		
				Acc.	Sens.	Spec.	Acc.	Sens.	Spec.
[1,2,3,5]	(1,000;1,000)	0.64	0.02	0.94	0.96	0.90	0.96	1.00	0.88
	(1,000;2,500)			0.94	0.96	0.90	0.96	1.00	0.87
	(10,000;1,000)			0.95	0.97	0.90	0.95	0.98	0.89
	(10,000;2.500)			0.95	0.97	0.90	0.95	0.98	0.89
[6,7,8]	(1,000;1,000)	0.03	0.05	0.93	0.05	1.00	0.98	0.99	0.98
	(1,000;2,500)			0.93	0.01	1.00	0.98	0.99	0.98
	(10,000;1,000)			0.95	0.30	1.00	0.95	0.30	1.00
	(10,000;2.500)			0.95	0.29	1.00	0.95	0.29	1.00

Table 14: For eight settings with $p = 0.5$, rule profile $[1, 2, 3, 5]$ or $[6, 7, 8]$ and with different dimensions (N, P), RPA has been applied to 100 generated datasets. This table provides the corresponding accuracy, sensitivity and specificity.

rpf	(N, P)	Runtime (minutes)								
		Total			Finding clauses labelled 'G'			Finding clauses labelled 'A'		
		mean	min	max	mean	min	max	mean	min	max
[1,2,3,5]	(1,000;1,000)	0.7	0.5	0.9	0.3	0.3	0.4	0.4	0.2	0.6
	(1,000;2,500)	3.9	3.0	5.2	1.6	1.5	2.3	2.4	1.4	3.6
	(10,000;1,000)	5.2	2.5	14.9	2.6	2.4	5.7	2.6	0.0	9.2
	(10,000;2,500)	50.4	17.8	122.4	19.8	17.8	48.0	30.6	0.0	74.4
[6,7,8]	(1,000;1,000)	1.5	0.9	1.8	0.0	0.0	0.2	1.4	0.8	1.8
	(1,000;2,500)	11.2	7.8	14.5	0.1	0.0	1.7	11.1	6.2	14.5
	(10,000;1,000)	4.7	2.3	7.3	4.7	2.3	7.3	0.1	0.0	2.4
	(10,000;2,500)	34.1	17.6	52.3	33.1	17.6	52.0	1.0	0.0	17.3

Table 15: For eight settings with $p = 0.5$, rule profile $[1, 2, 3, 5]$ or $[6, 7, 8]$ and with different dimensions (N, P), RPA has been applied to 100 generated datasets. This table provides the corresponding accuracy, sensitivity and specificity. This table provides the corresponding runtimes.

3.4 Results for $p = 0.25$

Now we perform an identical set of experiments as in the previous Section 3.3, but with probability $p = 0.25$. Throughout all settings, generating clauses are found with the same or lower probability than for $p = 0.5$. Moreover, runtimes are considerably higher. The increased hardship that RPA endures can be explained with the degrees of overlap (cf. Section 3.1), which increase as p decreases. Apart from observing the general increased difficulty of finding generating clauses, one can derive similar insights as in the previous section. Hence, the following results are presented with no further elaboration.

rpf	(N, P)	Total runtime (minutes)					
		RPA			IRELAND		
		mean	min	max	mean	min	max
[1,2,3,5]	(1,000;1,000)	0.8	0.7	1.1	140	102	261
	(1,000;2,500)	3.9	3.6	4.5	113	83	146
	(10,000;1,000)	2.8	1.4	4.8	129	111	154
	(10,000;2,500)	26.7	13.2	43.3	117	88	156
[6,7,8]	(1,000;1,000)	1.1	0.9	1.4	94	72	118
	(1,000;2,500)	5.8	5.2	6.9	94	81	116
	(10,000;1,000)	3.0	2.1	4.0	100	85	131
	(10,000;2,500)	21.6	14.9	29.5	119	77	170

Table 16: To compare the runtimes of the algorithms RPA and IRELAND, we generated 10 datasets for eight settings with rule profile $[1, 2, 3, 5]$ or $[6, 7, 8]$, $p = 0.5$ and with different dimensions (N, P) and solved each dataset with both algorithms. This table reports the total runtimes.

rpf	(N, P)	Clause size	Of all generating clauses		Of all clauses labelled 'G'	Of all clauses labelled 'A'
			% Label 'G'	% Label 'A'	% Generating	% Generating
[1,2,3,5]	(1,000; 1,000)	1	1.00	0.00	1.00	N/A
		2	1.00	0.00	1.00	N/A
		3	0.65	0.12	0.97	0.08
		5	0.00	0.00	0.00	0.00
	(1,000; 2,500)	1	0.99	0.00	1.00	N/A
		2	1.00	0.00	1.00	N/A
		3	0.42	0.23	1.00	0.14
		5	0.00	0.00	0.00	N/A
	(10,000; 1,000)	1	1.00	0.00	1.00	N/A
		2	1.00	0.00	1.00	N/A
		3	0.99	0.00	1.00	N/A
		5	0.00	0.00	0.00	0.00
	(10,000; 2,500)	1	1.00	0.00	1.00	N/A
		2	1.00	0.00	1.00	N/A
		3	1.00	0.00	1.00	N/A
		5	0.00	0.00	N/A	0.00
[6,7,8]	(1,000; 1,000)	6	0.00	0.00	N/A	N/A
		7	0.00	0.00	N/A	N/A
		8	0.00	0.00	N/A	N/A
	(1,000; 2,500)	6	0.00	0.00	N/A	N/A
		7	0.00	0.00	N/A	N/A
		8	0.00	0.00	N/A	N/A
	(10,000; 1,000)	6	0.00	0.00	N/A	0.00
		7	0.00	0.00	N/A	0.00
		8	0.00	0.00	N/A	N/A
	(10,000; 2,500)	6	0.00	0.00	N/A	0.00
		7	0.00	0.00	N/A	0.00
		8	0.00	0.00	N/A	N/A

Table 17: For eight settings with $p = 0.25$, rule profile $[1, 2, 3, 5]$ or $[6, 7, 8]$ and with different dimensions (N, P), RPA has been applied to 100 generated datasets. The fraction of retrieved generating clauses is reported here.

rpf	(N, P)	Fraction of runs with labels 'G' before labels 'A'	Clauses labelled 'A' that are not generating					
			Number per dataset	Size	% Features contained in a generating clause			
			mean	mean	mean	min	med	max
[1,2,3,5]	(1,000; 1,000)	0.91	7.0	3.8	0.01	0.00	0.00	0.67
	(1,000; 2,500)	0.92	6.2	3.7	0.01	0.00	0.00	0.67
	(10,000; 1,000)	1.00	10.0	5.2	0.01	0.00	0.00	0.80
	(10,000; 2,500)	1.00	10.0	5.2	0.01	0.00	0.00	0.40
[6,7,8]	(1,000; 1,000)	0.93	9.4	3.9	0.02	0.00	0.00	0.50
	(1,000; 2,500)	0.91	8.1	3.9	0.01	0.00	0.00	0.33
	(10,000; 1,000)	1.00	10.0	5.4	0.03	0.00	0.00	0.40
	(10,000; 2,500)	1.00	10.0	5.3	0.01	0.00	0.00	0.40

Table 18: For eight settings with $p = 0.25$, rule profile $[1, 2, 3, 5]$ or $[6, 7, 8]$ and with different dimensions (N, P), RPA has been applied to 100 generated datasets. This table provides insight in the quality of the clauses labelled 'A'.

rpf	(N, P)	% Samples that are true cases	% Samples that are error cases	Using only clauses labelled 'G'			Using all found clauses		
				Acc.	Sens.	Spec.	Acc.	Sens.	Spec.
[1,2,3,5]	(1,000; 1,000)	0.30	0.03	0.95	0.89	0.98	0.98	1.00	0.97
	(1,000; 2,500)			0.95	0.88	0.98	0.98	1.00	0.97
	(10,000; 1,000)			0.95	0.89	0.98	0.95	0.91	0.97
	(10,000; 2.500)			0.95	0.89	0.98	0.95	0.91	0.97
[6,7,8]	(1,000; 1,000)	0.0003	0.0499	0.95	0.02	1.00	0.99	0.99	0.99
	(1,000; 2,500)			0.95	0.03	1.00	0.99	1.00	0.99
	(10,000; 1,000)			0.95	0.00	1.00	0.95	0.12	1.00
	(10,000; 2.500)			0.95	0.00	1.00	0.95	0.13	1.00

Table 19: For eight settings with $p = 0.5$, rule profile $[1, 2, 3, 5]$ or $[6, 7, 8]$ and with different dimensions (N, P), RPA has been applied to 100 generated datasets. This table provides the corresponding accuracy, sensitivity and specificity.

rpf	(N, P)	Runtime (minutes)								
		Total			Finding clauses labelled 'G'			Finding clauses labelled 'A'		
		mean	min	max	mean	min	max	mean	min	max
[1,2,3,5]	(1,000; 1,000)	1.2	0.9	1.4	0.3	0.2	0.4	0.9	0.5	1.2
	(1,000; 2,500)	7.2	5.0	11.0	1.6	0.9	3.8	5.6	3.3	9.0
	(10,000; 1,000)	17.1	15.7	17.8	3.0	2.8	3.2	14.0	12.9	14.7
	(10,000; 2,500)	135.5	122.9	148.6	23.6	21.5	30.0	111.9	101.2	123.3
[6,7,8]	(1,000; 1,000)	1.8	1.3	2.2	0.0	0.0	0.4	1.7	1.1	2.0
	(1,000; 2,500)	13.9	9.9	17.0	0.3	0.0	3.3	13.6	9.7	17.0
	(10,000; 1,000)	23.9	21.1	26.5	0.0	0.0	0.0	23.9	21.1	26.5
	(10,000; 2,500)	175.8	159.8	198.6	0.0	0.0	0.0	175.8	159.8	198.6

Table 20: For eight settings with $p = 0.5$, rule profile $[1, 2, 3, 5]$ or $[6, 7, 8]$ and with different dimensions (N, P), RPA has been applied to 100 generated datasets. This table provides the corresponding accuracy, sensitivity and specificity. This table provides the corresponding runtimes.

4. Discussion

This work proposes the *Remaining Positives Algorithm* (RPA) as a solution algorithm to a fundamental data science problem, namely to identify Boolean logic rules in DNF that classify samples based on binary or binarized features. RPA outputs a set of clauses, so that we can predict a dependent variable as well as provide the explicit input-output relationship. This paper introduces two easily computable Remaining Positives statistics to obtain axes on which features of interest separate themselves. This is used to navigate through the space of all clauses: the algorithm visits promising candidates and applies a classification procedure to decide whether we accept a given clause as generating. In many settings, RPA succeeds in finding the underlying generating clauses efficiently. Moreover, we introduce the Degree of Overlap concept that allows the user to anticipate the likelihood that RPA will detect the generating clauses contained in the underlying boolean rule, given a few simple characteristics of the dataset.

RPA is in many cases able to retrieve the true underlying Boolean rules in DNF from the data. It does so in a fraction of the time required by IRELAND, which is, to our knowledge, currently the fastest approach. The complexity of RPA increases only linearly in the number of samples N, a major advantage as a large number of samples is a requirement for retrieving complex relationships from data.

In some settings, RPA struggles to detect the underlying generating clauses, such as in the setting with rule profile $[6, 7, 8]$, with $p = 0.25$ and with error probability $\beta = 0.05$ (see Table 17). Here, not a single generating clause was retrieved, even with the number of rows as high as $N = 10,000$. This can be explained by the fact that in this setting, error cases vastly outnumber the true cases. The error probability $\beta = 0.05$ far exceeds the probabilities of the generating clauses of sizes 6, 7 and 8 being active, which are $(0.25)^6 = 0.0002$, $(0.25)^7 = 0.00006$ and $(0.25)^8 = 0.00002$, respectively. Consequently, the expected fraction of error cases is 0.0499 of all samples and the expected fraction of true cases is 0.0003 of all samples (see Table 19). Hence, one may anticipate that a dataset with $N = 10,000$ rows contains 499 cases that were caused by error and only 3 cases that were caused by the underlying boolean rule. Being unable to know which cases are the true cases, it is unreasonable to expect any algorithm to retrieve the boolean rule. Appropriate expectations are necessary.

To clarify the boundaries of what is possible, we recommend that future research develops formal notions for the feasibility of identifying the input-output relationships given certain characteristics of the data. The Degree of Overlap concept proposed in this paper is an initial approach, but it has not been thoroughly specified. Also, it is tailored to our framework of boolean rules in DNF, since it is defined in terms of the Remaining Positives statistics. One may wish to develop a more generic retrievability measure.

One may question the usefulness of the part of our algorithm that labels clauses as ambiguous. As observed in Section 3, finding clauses that are labelled 'A' can increase runtimes substantially and often leads to overfitting. However, this section has also shown that there are rare cases in which such clauses contain features of generating clauses. Investigating this further could lead to insightful future research.

Throughout this paper, we have assumed that the frequency p of occurences of a feature among samples is equal for all features. Also, we have assumed that there is independence

among features. However, in many applications such as in bioinformatics, these assumptions are not realistic. Before using RPA in such contexts, one should investigate to what extent this influences the algorithm's effectivity.

More than only a solution algorithm, this paper offers a flexible and transparent framework with many entry points for future adjustments and improvements. For example, for the version of RPA proposed in this paper, the generating domains are chosen such that both Remaining Positives statistics of a generating clause are in the generating domain with 0.999 probability. One could make the algorithm stricter or looser by changing this probability, hence by increasing or decreasing the size of the generating domain. This parameter thus allows the user to control the level of statistical support that is desired before accepting an AND clause as generating.

Our main motivation stems from genomics. Many diseases that we cannot cure today, such as Alzheimer's, cancer and ALS, have a - largely unknown - underlying genetic cause. Over the past decade major efforts have resulted in collecting genome data from patients with the studied disease as well as healthy controls. Analyzing these datasets holds the promise of identifying the genetic characteristics underlying the disease, which can lead to a better understanding of disease mechanisms and improved drug development. Boolean phrases in DNF are highly suitable for this purpose: both the dependent and independent data are binary or can be readily binarized, and the DNF form fits well with our understanding of how the underlying biological mechanisms could work (Knijnenburg et al., 2016).

Acknowledgments

This work was supported by the Netherlands Organization for Scientific Research (NWO) Veni grant VI.Veni.192.043.

References

Marleen Balvert. Iterative rule extension for logic analysis of data: an milp-based heuristic to derive interpretable binary classification from large datasets. arXiv preprint arXiv:2110.13664, 2021.

Allison Chang, Dimitris Bertsimas, and Cynthia Rudin. An integer optimization approach to associative classification. In Advances in neural information processing systems, pages 269–277, 2012.

William W Cohen. Fast effective rule induction. In Machine learning proceedings 1995, pages 115–123. Elsevier, 1995.

Sanjeeb Dash, Oktay Gunluk, and Dennis Wei. Boolean decision rules via column generation. Advances in neural information processing systems, 31, 2018.

Peter L Hammer and Tibérius O Bonates. Logical analysis of data—an overview: From combinatorial optimization to medical applications. Annals of Operations Research, 148 (1):203–225, 2006.

Theo A Knijnenburg, Gunnar W Klau, Francesco Iorio, Mathew J Garnett, Ultan McDermott, Ilya Shmulevich, and Lodewyk FA Wessels. Logic models to predict continuous outputs based on binary inputs with an application to personalized cancer therapy. Scientific reports, 6(1):1–14, 2016.

Dmitry Malioutov and Kush Varshney. Exact rule learning via boolean compressed sensing. In International conference on machine learning, pages 765–773. PMLR, 2013.

R Andrew McCallum and Kent A Spackman. Using genetic algorithms to learn disjunctive rules from examples. In Machine Learning Proceedings 1990, pages 149–152. Elsevier, 1990.

Ryszard S Michalski, Igor Mozetic, Jiarong Hong, and Nada Lavrac. The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In Proc. AAAI, volume 1986, pages 1–041, 1986.

J. Ross Quinlan. Learning logical definitions from relations. Machine learning, 5(3):239–266, 1990.

Ingo Ruczinski, Charles Kooperberg, and Michael LeBlanc. Logic regression. Journal of Computational and graphical Statistics, 12(3):475–511, 2003.

Mojtaba Seyedhosseini and Tolga Tasdizen. Disjunctive normal random forests. Pattern Recognition, 48(3):976–983, 2015.

Chuang Wu, Andrew S Walsh, and Roni Rosenfeld. Genotype phenotype mapping in rna viruses-disjunctive normal form learning. In Biocomputing 2011, pages 62–73. World Scientific, 2011.