Learning Horn Formulas From Positive and Negative Examples Over Chains by Offline and Online Algorithms

Miki Hermann
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, 91120 Palaiseau, France

Gernot Salzer
Technische Universität Wien, Vienna, Austria

Abstract

One of the core activities in learning is the synthesis of concepts and their relationships by generalizing from positive and negative examples. Information hidden in data becomes explicit, relations emerge that provide insights and serve as explanations. Nowadays, machine learning is able to process large quantities of data and to build models that classify new data with such a success that the algorithms are termed 'intelligent'. With most approaches, though, the models are black boxes: It is usually difficult to explain why a model classifies data in a particular way.

Our work is an approach to machine learning that provides explanations. We present algorithms that construct if-then rules (Horn clauses) from examples and counter-examples. We generalize results for binary data to finite totally-ordered domains, obviating the need to binarize data over such domains (which usually entails an increase in variables and output that is hard to interpret). We present both, an offline and an online algorithm. In the first case, the positive and negative examples are known from the outset, while in the second case the examples arrive one by one and lead to adaptions of the output. The offline algorithm turns out to be more efficient, with its runtime linear in the number of positive and negative examples and quadratic in the number of variables. Our C++ implementation of the algorithms is publicly available.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases Explainable AI, machine learning, Horn clauses, signed logic, finite domains, many-valued logics, offline algorithms, online algorithms.

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction and Summary of Results

In [7], Valiant introduced the concept of PAC-learning and focused attention on classes of Boolean formulas learnable in polynomial time. Angluin, Frazier, and Pitt presented a polynomial-time AFP-algorithm for learning Horn formulas [1], and Arias and Balcázar proved its optimality [2].

The recent revival of machine learning, triggered by the flood of *big data*, sparked a new interest in learning formulas, either without any restriction or formulas satisfying some syntactic requirements. Among them are formulas in conjunctive normal (CNF) form, in disjunctive normal form, or CNF formulas with a limited number of literals per clause (k-CNF). Horn formulas, i.e., CNFs with at most one positive literal per clause, are of particular interest as they can be understood as if-then rules and may serve as an explanation accessible also to non-logicians. Moreover, the satisfiability of Horn formulas can be checked in polynomial time, their models (satisfying assignments) form a semi-lattice, with the minimal one easily computable.

Learning Boolean formulas from data requires its *binarization*, i.e., the data values have to be mapped from the original domain to Boolean vectors. Each component of the vector

corresponds to a two-valued variable in the learned formula. See e.g. the MCP system [5] for an implementation of this approach. Binarization is problematic for several reasons. When choosing a compact representation, the number of Boolean variables is kept to a minimum, but the structure inherent in the original domain gets lost and is inaccessible for learning the formula. Moreover, interpreting the learned formula in the original context will most likely be difficult. On the other hand, when using a binary encoding that respects the original structure, the number of required Boolean variables quickly increases to a point where the construction of the formula becomes infeasible.

One way to avoid binarization is to extend propositional logic by atomic formulas with an inherent structure. Atomic formulas are no longer atomic in a strict sense, but their interaction with the propositional level is kept to a minimum. An example of such an extension is signed logic, where inequalities of the form $x \geq d$ and $x \leq d$, with x a numeric variable and d a domain value, take the role of propositional variables. See [3, 6] for surveys on reasoning in many-valued logics via signed logic. This approach keeps the simplicity of propositional logic, but obviates the need for binarization in the case of totally ordered domains.

For finite, totally ordered domains, Gil et al. [4] present a polynomial-time algorithm for constructing a Horn formula φ in signed logic such that each assignment in a given set M satisfies φ, while the number of assignments falsifying φ is maximized. In this paper, we study a similar problem: Given two nonempty sets T and F of positive and negative examples, respectively, the task is to construct a Horn formula φ in signed logic such that every positive example satisfies φ and every negative example falsifies φ. We intentionally remain vague about examples neither in T nor in F. We study both the *offline* case, where T and F are supplied upfront, and the *online* case, where the examples, positive or negative, arrive one by one.

2 Preliminaries

The following definitions are an excerpt from [4]. Let $D = \{0, \ldots, n-1\}$ be a finite totally ordered domain, called a *chain*, of cardinality n and let V be a countably infinite set of variables. For a variable $x \in V$ and a value $d \in D$, the inequalities $x \geq d$ and $x \leq d$ are called positive and negative *literal*, respectively. The set of *formulas* over D and V is defined as follows:

- the logical constants *false* and *true* are formulas;
- literals are formulas;
- if φ and ψ are formulas, then the expressions $(\varphi \wedge \psi)$ and $(\varphi \vee \psi)$ are formulas.

We write $\varphi(x_0, \ldots, x_{\ell-1})$ to indicate that formula φ contains exactly the variables $x_0, \ldots, x_{\ell-1}$. Note that literals require space $O(\log n)$. Indeed, since d is bounded by n, its binary coding has length $O(\log n)$.

A *clause* is a disjunction of literals. It is a *Horn* clause if it contains at most one positive literal. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. It is a Horn formula if it is a conjunction of Horn clauses. Since all formulas in the sequel are in conjunctive normal form, we will use the term 'formula' as a synonym for CNF formula.

Note that contrary to the Boolean case, a clause in our formalism can contain the same variable twice without being reducible, if the variable occurs both in a positive and in a negative literal. However, variables can be assumed to occur only once per literal type, as $x \geq d \vee x \geq d'$ can be reduced to $x \geq \min(d, d')$ and dually, $x \leq d \vee x \leq d'$ can be reduced to $x \leq \max(d, d')$.

An *assignment* for a formula $\varphi(x_0, \ldots, x_{\ell-1})$ is a mapping $m\colon \{x_0, \ldots, x_{\ell-1}\} \to D$ assigning a domain element to each variable. The *satisfaction relation* $m \models \varphi$ is inductively defined as follows:

1. $m \models \mathit{true}$ and $m \not\models \mathit{false}$;
2. $m \models x \le d$ if $m(x) \le d$, and $m \models x \ge d$ if $m(x) \ge d$;
3. $m \models \varphi \wedge \psi$ if $m \models \varphi$ and $m \models \psi$;
4. $m \models \varphi \vee \psi$ if $m \models \varphi$ or $m \models \psi$.

If m satisfies φ, it is called a *model* of φ. We extend the satisfaction relation to sets of assignments by defining $M \models \varphi$ if $m \models \varphi$ holds for all $m \in M$. If we arrange the variables in some arbitrary but fixed order, say $x_0, \ldots, x_{\ell-1}$, then the assignments can be identified with the vectors in D^ℓ. The j-th component of a vector m, denoted by $m[j]$, gives the value of the j-th variable, i.e., $m[j] = m(x_j)$.

We extend the total orderings $\le$ and $\ge$ on the domain D to partial orderings $\preceq$ and $\succeq$ on vectors $m, m' \in D^\ell$ as follows:

1. $m \preceq m'$ if $m[j] \le m'[j]$ holds for all $j = 0, \ldots, \ell - 1$;
2. $m \succeq m'$ if $m[j] \ge m'[j]$ holds for all $j = 0, \ldots, \ell - 1$.

Likewise, the minimum operator is extended to vectors by setting

$$\min(m, m') = (\min(m[0], m'[0]), \ldots, \min(m[\ell-1], m'[\ell-1])).$$

The problem of learning a Horn formula from positive and negative examples, which we study in this paper, is defined as follows.

Problem: LEARN_HORN_FORMULA
Input: Nonempty sets of negative examples F and positive examples T ($\emptyset \subset F, T \subseteq D^\ell$).
Output: Horn formula φ such that $T \models \varphi$ and $f \not\models \varphi$ holds for each $f \in F$.

3 Horn Closure

The Horn closure $\mathrm{HC}(M)$ of a set of vectors $M \subseteq D^\ell$ is the smallest set containing M and being closed under minimum. Formally, $\mathrm{HC}(M)$ is the smallest set such that $M \subseteq \mathrm{HC}(M)$ and $\min(m, m') \in \mathrm{HC}(M)$ if $m, m' \in \mathrm{HC}(M)$. It is well-known (and stated in [4]) that M is the set of all models of a Horn formula φ if and only if M is closed under minimum, i.e., if $M = \mathrm{HC}(M)$. Hence, for Horn formulas φ, the property $T \models \varphi$ is equivalent to $\mathrm{HC}(T) \models \varphi$.

The cardinality of the Horn closure $\mathrm{HC}(M)$ can be exponentially larger than the one of the original set of vectors M, as the following example shows. For a fixed parameter $p \in \{0, \ldots, \ell - 1\}$, let $M_p = \{m_0^p, \ldots, m_{n-2}^p\}$ be a set of vectors from D^ℓ, where

$$m_i^p[j] = \begin{cases} i & \text{if } j = p \\ n-1 & \text{otherwise} \end{cases}$$

Each set M_p contains $n-1$ vectors of arity ℓ. The vectors differ only at the position p, taking the values 0 to $n-2$. All other positions of the vectors are equal to $n-1$.

Now let $M = M_0 \cup \cdots \cup M_{\ell-1}$ be the union of these sets M_p; its cardinality is $|M| = \ell(n-1)$. As it can easily be checked, the Horn closure is given by $\mathrm{HC}(M) = D^\ell \setminus \{(n-1, \ldots, n-1)\}$, a set of cardinality $n^\ell - 1$, which is exponentially larger than the original set M. Therefore the lower bound for computing the Horn closure $\mathrm{HC}(M)$ of a set of vectors $M \subseteq D^\ell$ is $\Omega(n^\ell)$. This also implies that in the worst case, Algorithm 1 has to run in exponential time.

The computation of a Horn closure $\mathrm{HC}(M)$ of a set of vectors $M \subseteq D^\ell$ is in fact the computation of the corresponding meet-semilattice $\langle M, \wedge \rangle$, where the meet operation $\wedge$ is

Algorithm 1 Horn Closure

Input: Set of vectors $M \subseteq D^\ell$.
Output: Horn closure $\mathrm{HC}(M)$.

```
1:  function HORN_CLOSURE(M)
2:      H ← ∅
3:      M' ← M
4:      while M' ≠ ∅ do
5:          let m' ∈ M'
6:          for all m ∈ M do
7:              m'' ← min(m', m)
8:              if m'' ∉ H then
9:                  M' ← M' ∪ {m''}
10:         H ← H ∪ {m'}
11:         M' ← M' ∖ {m'}
12:     return H
```

the minimum operator extended to vectors. Since the minimum operator is associative, commutative, and idempotent, each new vector m'' inserted into M' on line 9 can be seen as a sequence of vectors $m_1, \ldots, m_k$ from the original input set M produced by a successive application of min: $m'' = (\cdots((m_1 \wedge m_2) \wedge m_3) \cdots \wedge m_k)$. Therefore it is sufficient to apply the inner for-loop on line 6 only through the vectors from the original set M. The set M' is initiated by the original input vectors from M and throughout the outer while-loop on line 4 receives all vectors produced from M by successive application of the minimum operation until the saturation of the output set H. Since there are at most n^ℓ vectors of arity ℓ over a finite domain of cardinality n, the intermediate set M' cannot receive more than n^ℓ vectors and therefore the while-loop will be executed at most n^ℓ times. This proves that Algorithm 1 runs in time $O(|M|\, n^\ell)$.

4 Ensuring Disjoint Sets

It is a logic requirement to have the sets of positive and negative examples disjoint. However, in the case of learning Horn formulas from positive and negative examples, we must have a stronger requirement. Since the models of a Horn formula are closed under the operation of minimum, we must require the set of negative examples F to be disjoint from the Horn closure $\mathrm{HC}(T)$ of positive examples T. The straightforward approach would consist of computing the Horn closure $\mathrm{HC}(T)$ first, followed by the test of an empty intersection $\mathrm{HC}(T) \cap F$. Since the Horn closure $\mathrm{HC}(T)$ can be exponentially larger than the original set of positive examples T, as we saw before, this approach is not viable. We need another way of deciding if a negative example $f \in F$ belongs to the Horn closure $\mathrm{HC}(T)$ without computing it. For this, we will make use of the property that a Horn closure is closed under the minimum operation, as well as of the fact that we can compute the minimum operation of a set of vectors in polynomial time.

Every negative example $f \in F$ can be present in the Horn closure $\mathrm{HC}(T)$ only as a result of a minimum closure of vectors $t \in T$ which satisfy the relation $t \succeq f$. Hence, we need first to compute the subset of positive examples $T_{\succeq f} = \{t \in T \mid t \succeq f\}$ which are bigger or equal than f, followed by the computation of its minimum vector $m_f = \min T_{\succeq f}$.

▶ **Proposition 1.** *Let T and F be the sets of positive and negative examples of arity ℓ. Let $T_{\succeq a} = \{t \in T \mid t \succeq a\}$ be the subset of positive examples bigger or equal to a vector $a \in D^\ell$. A negative example $f \in F$ belongs to the Horn closure* $\mathrm{HC}(T)$ *if and only if $f = \min T_{\succeq f}$.*

Proof. If $f = \min T_{\succeq f}$ holds, then f is either already present in T or a result of a minimum closure of the subset $T_{\succeq f} \subseteq T$. In both cases it implies that $f \in \mathrm{HC}(T)$ holds.

If a negative example $f \in F$ belongs to the Horn closure $\mathrm{HC}(T)$, then either $f \in T$ holds or f is a minimum closure of a subset $T' \subseteq T$. The subset T' cannot contain a vector which is smaller than f or incomparable with f, since then the minimum closure of T' would be smaller than f. Hence the subset T' must satisfy the relation $T' \subseteq T_{\succeq f}$. Suppose now that T' is maximal, $f = \min T'$, and $f \neq \min T_{\succeq f}$. This implies that there exists a vector $t \in T_{\succeq f} \setminus T'$. But this indicates that we can compute another vector $f' = \min(f, t)$ which is strictly smaller than f. This is a contradiction with the maximality of the set T'. Therefore the equality $f = \min T_{\succeq f}$ must be satisfied. ◀

The presence of a negative example $f \in F$ in the Horn closure $\mathrm{HC}(T)$ is computed by Algorithm 2.

Algorithm 2 Is a given vector in the Horn closure of a vector set?

Input: Vector $f \in F$ and a set of vectors T.
Output: Boolean value *true* if $f \in \mathrm{HC}(T)$, *false* otherwise.

```
1: function IN_HORN_CLOSURE(f, T)
2:     m_{⪰f} ← (n − 1, ..., n − 1)
3:     for all t ∈ T do
4:         if t ⪰ f then
5:             m_{⪰f} ← min(t, m_{⪰f})
6:     if f = m_{⪰f} then
7:         return true
8:     else
9:         return false
```

▶ **Proposition 2.** *Let f be a vector and T a set of vectors, both of arity ℓ. Algorithm 2 determines if the vector f belongs to the Horn closure* $\mathrm{HC}(T)$ *in time $O(|T|\,\ell \log n)$.*

Proof. Algorithm 2 performs a loop for each $t \in T$. Within this loop it compares vectors t and f and computes the minimum of t and $m_{\succeq f}$, both in time $O(\ell \log n)$. Finally it compares vectors f and $m_{\succeq f}$ for equality in time $O(\ell \log n)$. ◀

▶ **Corollary 3.** *Let T and F be sets of positive and negative examples of arity ℓ. The property whether a set of negative examples F is disjoint from the Horn closure* $\mathrm{HC}(T)$ *can be determined in time $O(|F|\,|T|\,\ell \log n)$.*

5 Learning Horn Formulas Offline

Suppose first that both sets of positive examples T and negative examples F are given upfront, before the start of the learning process.

If we were interested to construct an arbitrary CNF formula φ falsifying only the negative examples F, for each vector $f \in F$, where $f = (f_0, \ldots, f_{\ell-1})$, we would construct a clause $c_f = c_0^f \vee \cdots \vee c_{\ell-1}^f$, such that each subclause c_i^f is equal to a disjunction of one negative and one

Algorithm 3 Learn Horn Formula

Input: Nonempty vector sets of negative examples F and positive examples T of arity ℓ.
Output: Horn formula $\varphi(x_0, \ldots, x_{\ell-1})$, such that $\mathrm{HC}(T) \models \varphi$ and $f \not\models \varphi$ for each $f \in F$.

function LEARN_HORN_FORMULA(F, T)
  $\varphi \leftarrow \top$ ▷ Initiate Horn formula φ to *true*.
  for all $f \in F$ **do** ▷ $f = (f_0, \ldots, f_{\ell-1})$
    if IN_HORN_CLOSURE$(f, T) =$ *true* **then** ▷ See Algorithm 2
      return $\bot$
    $c \leftarrow \bot$ ▷ Initiate empty Horn clause c.
    for $i \leftarrow 0$ **to** $\ell - 1$ **do** ▷ For each coordinate i ...
      if $f_i > 0$ **then**
        $c \leftarrow c \vee (x_i \leq f_i - 1)$ ▷ ... add the negative literal.
    if $T \models c$ **then** ▷ If Horn clause c is satisfied by all positive examples T...
      $\varphi \leftarrow \varphi \wedge c$ ▷ ... then add c to the Horn formula φ.
    else
      eliminated $\leftarrow$ *false*
      $i \leftarrow 0$
      while $\neg$*eliminated* **do** ▷ Search for suitable positive literal.
        if $f_i < n - 1$ **then**
          $c' \leftarrow c \vee (x_i \geq f_i + 1)$ ▷ Construct definite Horn clause c'.
          if $T \models c'$ **then** ▷ If definite Horn clause c' is satisfied by T ...
            $\varphi \leftarrow \varphi \wedge c'$ ▷ ... then add c' to the Horn formula φ ...
            eliminated $\leftarrow$ *true* ▷ ... and exit the while-loop.
        $i \leftarrow i + 1$
  return φ

positive literal, which together eliminate exactly the value f_i: $c_i^f = (x_i \leq f_i - 1) \vee (x_i \geq f_i + 1)$. Obviously, the negative literal is dropped if $f_i = 0$ and the positive literal is dropped if $f_i = n - 1$. The resulting CNF formula φ is the conjunction of all clauses c_f for each negative example $f \in F$. However, this procedure does not produce a Horn formula in general, since it contains $|F|$ positive literals in each clause, whereas at most one is allowed in a Horn clause. We need to choose among these positive literals only one, if necessary. To achieve this, we will exploit the closure property with respect to the minimum operation of $\mathrm{HC}(T)$, as well as the possibility to test if the sets $\mathrm{HC}(T)$ and F are disjoint, without the construction of the Horn closure.

Algorithm 3 construct one Horn clause for each negative example f in F. For each negative example $f = (f_0, \ldots, f_{\ell-1})$ it constructs first a prototype c of a Horn clause by the for-loop in lines 7 to 9, where it supplies one negative literal $x_i \leq f_i - 1$ for each coordinate $i = 0, \ldots, \ell - 1$, provided that $f_i > 0$. At the end of the for-loop, the prototype clause c already falsifies the negative example f, as it was required, but we are not sure if it satisfies all positive examples in T. To ensure this property, Algorithm 3 performs a satisfiability test on line 10. If this test succeeds, the clause c is added to the formula φ and the main for-loop proceeds with the next negative example from the set F. If this test fails, Algorithm 3 continues to search for a positive literal which will be added to the clause c. This is done within the while-loop on lines 15 to 21. Algorithm 3 searches for the first possible positive literal $x_i \geq f_i + 1$ to be added to c, forming a new definite Horn clause c', which is then tested on line 18 if it is satisfied by all positive examples from T. Such positive literal always exists.

▶ **Lemma 4.** *Let F and T be sets of negative and positive examples, both of arity ℓ, such that $\mathrm{HC}(T) \cap F = \emptyset$. Algorithm 3 within the while-loop on lines 15 to 21 always finds a definite Horn clause c' which is satisfied by all positive examples T.*

Proof. Suppose that there exists a negative example $f = (f_0, \ldots, f_{\ell-1})$, for which the clause c' is falsified by a subset T' of positive examples from T. Since both T and T' are supposed to be sets of models for a Horn formula, they are closed under the minimum operation. Let $t^* = \min T'$ be the minimum closure of T', where $t^* = (t_0^*, \ldots, t_{\ell-1}^*)$. Moreover the relation $t' \succeq t^*$ holds for all vectors $t' \in T'$. If all vectors from T' falsify the clause c', also the vector t^* must falsify that clause. This means that for each $i = 0, \ldots, \ell - 1$, we have $t_i^* \geq f_i$ and there exists a position j, such that $t_j^* \leq f_j$. This implies $t_j^* = f_j$ and the relation $t^* \succeq f$ holds. There must be another position $k \neq j$ for which we have $t_k^* > f_k$, otherwise the vectors t^* and f would be equal, but this would violate the disjointedness condition $\mathrm{HC}(T) \cap F = \emptyset$. But then there exists a clause $c'' = (x_0 \leq f_0 - 1) \vee \cdots \vee (x_{\ell-1} \leq f_{\ell-1} - 1) \vee (x_k \geq f_k + 1)$, which is falsified by the negative example f and satisfied by the positive example t^* and therefore also by all vectors from T'. Since k is one of the coordinate positions, Algorithm 3 will find it within the while-loop on lines 15 to 21. This is a contradiction with the fact that t^* falsifies the clause c'. ◀

▶ **Theorem 5.** *Let F and T be sets of negative and positive examples, both of arity ℓ. Algorithm 3 constructs a Horn formula φ, such that $\mathrm{HC}(T) \models \varphi$ and $f \not\models \varphi$ for each $f \in F$ in time $O(|F|\,|T|\,\ell^2 \log n)$.*

Proof. The correctness of Algorithm 3 follows from the analysis before Lemma 4.

The test on line 4 takes time $O(|F|\,|T|\,\ell \log n)$ according to Corollary 3. The for-loop constructing the prototype clause c on lines 7 to 9 takes time $O(\ell \log n)$. The test on line 10 takes time $O(|T|\,\ell \log n)$. The while-loop on lines 15 to 21 is executed at most ℓ-times. The

tests on lines 15 and 16 take time $O(\log n)$. The test on line 18 takes time $O(|T|\,\ell \log n)$. The main loop of the algorithm is executed $|F|$-times. Therefore the resulting time of Algorithm 3 is $O(|F|\,|T|\,\ell^2 \log n)$ in the worst case. ◀

6 Learning Horn Formulas Online

Suppose now that the vectors are supplied one at a time, just with the indication whether it is a positive or a negative example. How can we modify Algorithm 3 to adapt it for an online process?

Algorithm 4 Learn Horn Formula Online

Input: Vectors $m \in D^\ell$ of arity ℓ presented on input one at a time.
Output: Horn formula $\varphi(x_0, \ldots, x_{\ell-1})$, such that $\mathrm{HC}(T) \models \varphi$ and $F \not\models \varphi$.

```
1:  function LEARN_HORN_ONLINE( )
2:      T ← F ← pivot ← origin ← guard ← ∅
3:      φ ← ⊤
4:      while m ← READ(input) do
5:          if example(m) = positive & m ∉ T then
6:              T ← T ∪ {m}
7:              φ ← LEARN_POSITIVE(m, φ, F, T, pivot, origin, guard)
8:          else if example(m) = negative & m ∉ F then
9:              F ← F ∪ {m}
10:             φ ← LEARN_NEGATIVE(m, φ, F, T, pivot, origin, guard)
11:     return φ
```

The immediate first observation is that we cannot compute the sets $T_{\succeq f}$ for each negative example $f \in F$, since we do not know the sets of positive and negative examples upfront. Even if we already got a particular negative example f, its guarding set $T_{\succeq f}$ can evolve when other positive examples arrive later. For each negative example, the set $T_{\succeq f}$ serves as a *guard* to prevent mixing negative examples F with the Horn closure $\mathrm{HC}(T)$. It is also out of the question to recompute $T_{\succeq f}$ from scratch when a new positive example has been read from input. We do not need to keep the whole set $T_{\succeq f}$, but only its minimal closure vector $\min T_{\succeq f}$. This will be our *guard* for each negative example $f \in F$. when a new negative example f has been read from the input, we first initiate its $\mathit{guard}[f]$ to a non-existent dummy vector $(n, \ldots, n)$, which is bigger than any possible positive or negative example, then compute its minimal closure with all already known positive examples $t \in T$ satisfying the relation $t \succeq f$. Finally we compare the negative example f with $\mathit{guard}[f]$. The equality between them indicates the presence of f in the Horn closure $\mathrm{HC}(T)$ of positive examples T read up to now. When a new positive example t arrives, compare it to every negative example $f \in F$ read until now. If the relation $t \succeq f$ holds, we update the $\mathit{guard}[f]$ and compare it with f. The equality $\mathit{guard}[f] = f$ indicates the presence of f in the Horn closure $\mathrm{HC}(T)$. Note that the arrival of a new positive example t can change the Horn closure $\mathrm{HC}(T)$ tremendously.

Another observation is that a new positive example t can falsify several clauses of an already constructed Horn formula φ. If the Horn closure $\mathrm{HC}(T)$ is disjoint from F, where T and F are sets of positive and negative examples read from input until now, which is verified by means of the $\mathit{guard}[f]$ for each $f \in F$, we know that there exists another Horn formula φ' similar to φ, where the clauses from φ falsified by t are repaired. Each negative example f triggers the construction of a new clause c. To perform the repairs, we must memorize the

Algorithm 5 Learn Online from Negative Example

function LEARN_NEGATIVE($f, \varphi, F, T,$ **addr** *pivot*, **addr** *origin*, **addr** *guard*)
  $guard[f] = (n, \ldots, n)$ ▷ Initiate $guard[f]$ to non-existent big vector.
  for all $t \in T$ **do** ▷ Compute $\min T_{\succeq f}$ equal to $guard[f]$.
    if $t \succeq f$ **then**
      $guard[f] \leftarrow \min(t, guard[f])$
  if $guard[f] = f$ **then** ▷ If $f \in \mathrm{HC}(T)$ (see Proposition 1) ...
    return $\bot$ ▷ ... then no Horn formula possible.
  $c \leftarrow \bot$ ▷ Initiate empty Horn clause c.
  for $i \leftarrow 0$ **to** $\ell - 1$ **do** ▷ For each coordinate i ...
    if $f_i > 0$ **then**
      $c \leftarrow c \vee (x_i \leq f_i - 1)$ ▷ ... add the negative literal.
  if $T \models c$ **then** ▷ If Horn clause c is satisfied by all positive examples T...
    $\varphi \leftarrow \varphi \wedge c$ ▷ ... then add c to the Horn formula φ.
    $pivot[c] \leftarrow -1$ ▷ There is no *pivot*, i.e., no positive literal.
    $origin[c] \leftarrow f$ ▷ Mark f as the origin of the clause c.
  else
    $eliminated \leftarrow false$
    $i \leftarrow 0$
    while $\neg eliminated$ **do** ▷ Search for suitable positive literal.
      if $f_i < n - 1$ **then**
        $c' \leftarrow c \vee (x_i \geq f_i + 1)$ ▷ Construct definite Horn clause c'.
        if $T \models c'$ **then** ▷ If definite Horn clause c' is satisfied by T ...
          $\varphi \leftarrow \varphi \wedge c'$ ▷ ... then add c' to the Horn formula φ.
          $pivot[c'] \leftarrow i$ ▷ Mark the *pivot* coordinate of positive literal in c' ...
          $eliminated \leftarrow true$ ▷ ... and exit the while-loop.
      $i \leftarrow i + 1$
    $origin[c'] \leftarrow f$ ▷ Mark f as the origin of the clause c'.
  return φ

Algorithm 6 Learn Online from Positive Example

```
1: function LEARN_POSITIVE(t, φ, F, T, addr pivot, origin, addr guard)
2:     for all f ∈ F do                ▷ Update guard for every received negative example.
3:         if f ⪰ t then
4:             guard[f] ← min(t, guard[f])
5:             if guard[f] = f then                ▷ If f ∈ HC(T) (see Proposition 1) ...
6:                 return ⊥                        ▷ ... then no Horn formula possible.
7:     for all c ∈ φ do                            ▷ Check every clause c in formula φ.
8:         if t ⊭ c then                           ▷ If new positive example falsifies c ...
9:             f ← origin[c]                       ▷ f = (f_0, ..., f_{ℓ-1}).
10:            i ← pivot[c]
11:            if i ≥ 0 then
12:                c ← c ∖ (x_i ≥ f_i + 1)         ▷ ... then remove the positive literal.
13:            eliminated ← false
14:            while ¬eliminated do                ▷ Search for new positive literal.
15:                i ← i + 1
16:                if f_i < n − 1 then
17:                    c′ ← c ∨ (x_i ≥ f_i + 1)    ▷ Construct new clause prototype c′.
18:                    if T ⊨ c′ then              ▷ If prototype c′ satisfies all positive examples ...
19:                        c ← c′                  ▷ ... then replace the old clause c by new prototype c′.
20:                        pivot[c] ← i
21:                        eliminated ← true
22:    return φ
```

origin of the clause c, i.e., we must set $\mathit{origin}[c] = f$. The only part of a clause c which can be repaired is its positive literal. Hence we need to memorize the *pivot* coordinate i of the negative example f, which added the positive literal $(x_i \geq f_i)$ to c, i.e., we set $\mathit{pivot}[c] = i$. If the clause c has no positive literal, its *pivot* is set to -1. The repair is performed as follows. First the original positive literal (if any) is removed. Then we search for a new positive literal $(x_j \geq f_j)$ starting from the coordinate $j = \mathit{pivot}[c] + 1$. There is no need to search for a new positive literal at coordinates smaller than $\mathit{pivot}[c]$ since they have been eliminated already before. Since the sets $\mathrm{HC}(T)$ and F are disjoint, we are guaranteed to find another coordinate which generates a suitable positive literal.

The aforementioned developments are summarized in online Algorithms 5 and 6. Algorithm 5 learns from negative examples, whereas Algorithm 6 from positive ones. Algorithm 5 is only a slight modification of Algorithm 3. Algorithm 6 copies the part of Algorithm 3 for constructing the positive literal. The overhead Algorithm 4 just serves as a common wrap for calling the subsequent online algorithms for learning from positive and negative examples, as well as a store for global variables for sets of positive and negative examples, and also for the three structures *guard*, *origin*, and *pivot*. Since the last three structures must be globally updated, the online learning algorithms receive and submit them by their address, denoted by the keyword **addr** , as it is done e.g. in C++ by the operator `&`.

▶ **Theorem 6.** *Let F and T be the gathered sets of negative and positive examples, respectively. Let φ be the constructed Horn formula, such that $\mathrm{HC}(T) \models \varphi$ and $f \not\models \varphi$ hold for each $f \in F$.*

- *If f_+ is a new negative example then Algorithm 5 learns a new Horn formula φ', such that $\mathrm{HC}(T) \models \varphi'$ and $f' \not\models \varphi'$ for each $f' \in F \cup \{f_+\}$, in time $O(|T|\,\ell^2 \log n)$.*
- *If t_+ is a new positive example then Algorithm 6 learns a new Horn formula φ', such that*

$\mathrm{HC}(T \cup \{t_+\}) \models \varphi'$ *and* $f \not\models \varphi'$ *for each* $f \in F$, *in time* $O(|F|\,|T|\,\ell^2 \log n)$.

Proof. The correctness of Algorithms 5 and 6 follows from the discussion before the statement of this theorem.

Algorithm 5: The for-loop on line 3 costs $O(|T|\,\ell \log n)$ time. Testing the condition on line 6 costs $O(\ell \log n)$ time. The for-loop on line 9 costs $O(\ell \log n)$ time. The satisfiability test on line 12 costs $O(|T|\,\ell \log n)$ time. The while-loop on line 19 is executed $\ell - 1$ times in the worst case. The satisfiability test on line 22 costs $O(|T|\,\ell \log n)$ time. This test is encapsulated in the while-loop on line 19, therefore the whole negative branch of the if-test from line 12 costs $O(|T|\,\ell^2 \log n)$ time, which dominates all other values.

Algorithm 6: The for-loop on line 2 is executed $|F|$ times. Each execution costs $O(\ell \log n)$ time. The number of clauses in the formula φ is bounded by the cardinality $|F|$ of the negative examples, therefore the for-loop on line 7 is executed $|F|$ times in the worst case. The test on line 8 costs $O(\ell \log n)$ time. The while-loop on line 14 is executed ℓ times in the worst case. The satisfiability testing on line 18 costs $O(|T|\,\ell \log n)$ time. Considering the encapsulations, the overall time for execution of the for-loop on line 7 is $O(|F|\,|T|\,\ell^2 \log n)$. ◀

When we compare the offline and online algorithms for learning Horn formulas, we see that the online algorithm is more costly than the offline one. The problem resides in the learning part from positive examples, whereas learning from negative examples is on average less costly than in the offline case. Let us analyze this situation in detail.

Suppose that F_* and T_* are the maximal sets of negative and positive examples, respectively, after all vectors have been read by the online algorithm. One learning step with a negative example costs time $O(|T|\,\ell^2 \log n)$, where $T \subseteq T_*$ is the set of already learned positive example until the current point. Hence, the overall cost of learning all negative examples is $O(|F_*|\,|T_*|\,\ell^2 \log n)$ in the worst case. This worst case occurs when all positive examples are learned first, before learning the negative examples and its asymptotic complexity corresponds exactly to the offline case. Of course, in the average case the factor $|T_*|$ will be reduced.

The real culprit for the bad performance is the learning of positive examples. Suppose that all negative examples have been learned first, followed by the process of learning all positive examples after them. Hence, the factor $|F|$ will always be equal to $|F_*|$. During the positive learning process the cardinality $|T|$ goes from 1 to $|T_*|$. After summing up these cardinalities, this triggers the overall time of positive learning equal to $O(|F_*|\,|T_*|^2\,\ell^2 \log n)$, which is bigger than that of the offline algorithm by a factor of $|T_*|$.

▶ **Corollary 7.** *Let* F *and* T *be sets of negative and positive examples, respectively, both of arity* ℓ. *The online Algorithm 4 constructs a Horn formula* φ, *such that* $\mathrm{HC}(T) \models \varphi$ *and* $f \not\models \varphi$ *hold for each* $f \in F$, *in overall time* $O(|F|\,|T|^2\,\ell^2 \log n)$.

7 Implementation

Both algorithms have been coded in C++ and are available from `github.com/miki-hermann/learn-horn`. The installation and use of the software is described in the file `README.md`. This software will be incorporated into a new version of MCP [5] to avoid exponential explosion caused by binarization.

8 Concluding Remarks

We presented offline and online algorithms for learning Horn formulas from positive and negative examples in multi-valued logic over chains. The online version of the learning algorithm turned out to perform worse in the worst case than the offline version.

References

1 Dana Angluin, Michael Frazier, and Leonard Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

2 Marta Arias and José L. Balcázar. Construction and learnability of canonical Horn formulas. *Machine Learning*, 85(3):273–297, 2011.

3 Matthias Baaz, Christian G. Fermüller, and Gernot Salzer. Automated deduction for many-valued logics. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1355–1402. Elsevier and MIT Press, 2001.

4 Àngel J. Gil, Miki Hermann, Gernot Salzer, and Bruno Zanuttini. Efficient algorithms for description problems over finite totally ordered domains. *SIAM Journal on Computing*, 38(3):922–945, 2008.

5 Miki Hermann and Gernot Salzer. MCP: capturing big data by satisfiability (tool description). In Chu-Min Li and Felip Manyà, editors, *Proceedings 24th International Conference on Theory and Applications of Satisfiability Testing (SAT 2021), Barcelona (Spain)*, volume 12831 of *Lecture Notes in Computer Science*, pages 207–215. Springer, 2021.

6 Reiner Hähnle and Gonzalo Escalada Imaz. Deduction in many-valued logics: a survey. *Mathware and Soft Computing*, 4(2):69–97, 1997.

7 Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.