The Prioritized Inductive Logic Programs

Shilong Ma\textsuperscript{1,2}, Yuefei Sui\textsuperscript{1,3} and Ke Xu\textsuperscript{1,2}

\textsuperscript{†} National Lab of Software Development Environment
Beihang University, Beijing 100083, China
\textsuperscript{‡} Key Laboratory of Intelligent Information Processing
Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China

Abstract: The limit behaviors of an inductive logic program should be an important research topic which has not been explored. An inductive logic program is \textit{convergent} if given an increasing sequence of example sets, the inductive logic program should produce a corresponding sequence of the Horn logic programs which has the set-theoretic limit; \textit{limit-correct} if the limit of the produced sequence of the Horn logic programs is correct with respect to the limit of the sequence of the example sets. We will show by examples that there exist incremental learning algorithms which are not limit-correct in some cases. A priority order is defined on the literals, and the prioritized version of the learning algorithm is proposed such that the prioritized version is limit-correct with respect to any increasing infinite sequence \{\textit{E}_k\} of the positive literal sets.

Keywords: Inductive Logic Program, Machine Learning, Limit.

1. Introduction

As information increases exponentially, it becomes important to discover useful knowledge in massive information. Inductive logic programming is used in learning a general theory from given examples. In incremental learning, the examples are usually given one by one. After a new example is obtained, the current theory learned from previous examples might need to be updated. Thus we get a sequence \{\textit{Π}_k\} of theories. Sometimes there are infinitely many examples so that this procedure does not stop, i.e., there is no natural number \textit{k} such that \textit{Π}_k = \textit{Π}_{k+1} = \cdots. For example, if the theories are assumed to be Horn logic programs, then there exists some Herbrand interpretation \textit{I} such that there is no finite Horn logic program \textit{Π} whose least Herbrand model is equal to \textit{I}, because the set of Herbrand interpretations is uncountable while the set of finite Horn logic programs is only countable ([10]). So we should consider using the limits of Horn logic programs to approximate these Herbrand interpretations.

The limit behaviors of an inductive logic program should be an important research topic which has not been explored. In computer science, most of the softwares and algorithms are incremental or online. The limit behaviors should be one aspect of the correctness of the softwares and algorithms. We consider incremental inductive logic programs. Assume that examples come in sequences, let \textit{E}_k be the example set at time \textit{k}. A learning algorithm \textit{A} produces a theory \textit{A}(\textit{E}_k) for every \textit{k}. We require that \textit{A} should satisfy the following requirements:

\textsuperscript{1}The project was partially supported by the National Natural Science Foundation of China and the National 973 Project of China under the grant number G1999032701.
\textsuperscript{2}Email: \{slma, kexu\}@nlsde.buaa.edu.cn.
\textsuperscript{3}Email: suiyyff@hotmail.com.
• the convergence: Given a sequence \( \{E_k\} \) of the example sets such that \( E_1 \subseteq E_2 \subseteq E_3 \subseteq \cdots \), the set-theoretic limit of \( \{A(E_k)\} \) exists;

• the limit-correctness: If \( A(E_k) \) is correct with respect to \( E_k \) for every \( k \) then the limit of \( \{A(E_k)\} \) should be correct with respect to the limit of \( \{E_k\} \), that is, \( \lim_{k \to \infty} A(E_k) \vdash e \) for any \( e \in \lim_{k \to \infty} E_k \).

In the following, our discussion is based on a fixed logical language which contains only finitely many constant, function and predicate symbols. Ma et al. ([7]) considered the limit behavior of the Horn logic programs, and proved the following theorem:

Theorem 1.1([7]). Given a sequence \( \{\Pi_n\} \) of Horn logic programs, if \( \Pi = \lim_{n \to \infty} \Pi_n \) exists and for every sufficiently large \( n \), \( \Pi_n \) satisfies an assumption that for every clause in \( \Pi_n \), every term occurring in the body also occurs in the head, then

\[
\lim_{n \to \infty} M(\Pi_n) = M(\lim_{n \to \infty} \Pi_n),
\]

where \( M \) is an operator such that for any Horn logic program \( \Pi \), \( M(\Pi) \) is the least Herbrand model of \( \Pi \).

To consider the limit-correctness of inductive logic programs, we assume that the inductive logic programs satisfy the convergence. Given an inductive logic program \( A \), if for every positive example set \( E_n \), \( A(E_n) \) is a Horn logic program which is correct with respect to \( E_n \), i.e., \( M(A(E_n)) \supseteq E_n \), and \( A(E_n) \) satisfies the assumption, then by theorem 1.1,

\[
M(\lim_{n \to \infty} A(E_n)) = \lim_{n \to \infty} M(A(E_n)) \supseteq \lim_{n \to \infty} E_n,
\]

so \( A \) is limit-correct. Hence, to make \( A \) satisfy the limit-correctness, we should design \( A \) such that for any input \( E \), \( A(E) \) is a Horn logic program satisfying the condition. We shall give an learning algorithm, called \( A \), for the incremental inductive logic programs, and two examples to show that \( A \) may not always produce a logic program satisfying the condition, hence, may not be limit-correct with respect to an infinite example set. We modify algorithm \( A \) to be a prioritized one, say \( B \), in which a priority order is defined on the literals. In detail, for any example set \( E \), let \( A(E) \) and \( B(E) \) be the Horn logic programs produced by \( A \) and the prioritized version \( B \) of \( A \), respectively. Then, for an infinite set \( E \) of positive examples, \( A(E) \) may not satisfy the condition, but \( B(E) \) satisfies the condition. Hence, \( B \) is limit-correct with respect to an infinite example set \( E \) on which the priority order is linear.

The paper is organized as follows. In section 2, we shall give some basic definitions (e.g. the distance on terms and formulas and the set-theoretic limits) and an learning algorithm \( A \). In section 3, we shall consider the limit-correctness of algorithm \( A \), and give two examples to show that \( A \) is not limit-correct and sensitive to the ordering of the examples. In section 4, we shall propose the prioritized version, say \( B \) of \( A \), and prove that \( B \) is limit-correct with respect to any increasing sequence of positive literals, and not sensitive to the ordering of the examples. The last section concludes the paper.

Our notation is standard, references are [1,2,6]. We use \( p, q \) to denote the literals, clauses or predicate symbols; \( l, l' \) to denote the literals; \( \pi_1, \pi_2 \) the clauses; and \( \Gamma, \Pi \) the sets of clauses (logic programs).

2. Basic definitions
In this section, we will first define a distance on on terms and formulas similar to Nienhuys-Cheng’s definition in [10], and then give a basic introduction to least general generalizations used in inductive learning.

**Definition 2.1.** Let \( f \) and \( g \) be an \( n \)-ary and an \( m \)-ary function symbols, respectively. The distance \( \rho \) is defined as follows.

1. \( \rho(t, t) = 0 \), for any term \( t \);
2. \( \rho(f(t_1, ..., t_n), g(s_1, ..., s_m)) = 1 \);
3. \( \rho(f(t_1, ..., t_n), f(s_1, ..., s_m)) = \frac{\max\{\rho(t_i, s_i) \mid 1 \leq i \leq n\}}{\max\{\rho(t_i, s_i) \mid 1 \leq i \leq n\} + 1} \),

where \( t_1, ..., t_n, s_1, ..., s_m \) are terms.

The above definition of the distance on terms is a little different from the one given by Nienhuys-Cheng in that the value which the distance can take has a simple form of \( \frac{1}{m} \) for some natural number \( m \). Such a distance is used to define the distance between two trees in graph theory. Every term \( t \) can be taken as a tree \( T_t \). For example, \( t = f(t_1, ..., t_n) \), the tree \( T_t \) has a root with symbol \( f \) and \( n \)-many children \( T_{t_1}, ..., T_{t_n} \). We say that two terms \( t \) and \( t' \) are the same to depth \( m \), i.e. \( \rho(t, t') = \frac{1}{m} \) if \( T_t \) and \( T_{t'} \) are the same to depth \( m \). The distance defined here have the basic properties that the distance defined by Nienhuys-Cheng has.

Given two clauses \( \pi_1, \pi_2 \), to compute the least general generalization of \( \pi_1 \) and \( \pi_2 \), denoted by \( \text{lgg}(\pi_1, \pi_2) \), we give the following procedure:

1. Given two terms \( t = f(t_1, ..., t_n) \) and \( s = g(s_1, ..., s_m) \), we give the following procedure:

   \[
   \text{lgg}(t, s) = \begin{cases} 
   v & \text{if } f \neq g \\
   f(\text{lgg}(t_1, s_1), ..., \text{lgg}(t_n, s_m)) & \text{if } f = g,
   \end{cases}
   \]

   where \( v \) is any new variable.

2. Given two literals \( l_1 = (\neg)^k p(t_1, ..., t_n) \) and \( l_2 = (\neg)^q q(s_1, ..., s_m) \),

   \[
   \text{lgg}(l_1, l_2) = \begin{cases} 
   \text{undefined} & \text{if } k_1 \neq k_2 \text{ or } p \neq q \\
   (\neg)^{k_1} p(\text{lgg}(t_1, s_1), ..., \text{lgg}(t_n, s_n)) & \text{if } k_1 = k_2 \text{ and } p = q,
   \end{cases}
   \]

   where \( k_1 \) or \( 1, (\neg)^0 = , (\neg)^1 = \neg \).

3. Given two clauses \( \pi_1 = \{l_1, ..., l_n\} \) and \( \pi_2 = \{l_1', ..., l_m'\} \),

   \[
   \text{lgg}(\pi_1, \pi_2) = \{\text{lgg}(l_i, l_j') : 1 \leq i \leq n, 1 \leq j \leq m, \text{lgg}(l_i, l_j') \text{ is defined}\}.
   \]

To simplify the discussion, we assume that the literals mentioned below are ground.

**Definition 2.2.** Given a sequence \( \{E_k\} \) of sets of formulas, the set-theoretic limit of \( \{E_k\} \) exists, denoted by \( \lim_{k \to \infty} E_k \), if

\[
\lim_{k \to \infty} E_k = \text{lim}_{k \to \infty} E_k
\]

where

\[
\text{lim}_{k \to \infty} E_k = \{\phi : \exists k (\phi \in E_k)\}; \\
\text{lim}_{k \to \infty} E_k = \{\phi : \exists k_0 \forall k \geq k_0 (\phi \in E_k)\},
\]

where \( \exists k \) means that there are infinitely many \( k \).
We consider the incremental learning.

**Definition 2.3.** Given an incremental sequence \( \{E_k\} \) of literals, we say that an algorithm \( A \) is *convergent with respect to \( \{E_k\} \) if \( \lim_{k \to \infty} A(E_k) \) exists.

**Definition 2.4.** If \( A \) is convergent with respect to \( \{E_k\} \), then let \( E = \lim_{k \to \infty} E_k \) and \( A(E) = \lim_{k \to \infty} A(E_k) \), we say that \( A \) is *correct with respect to \( \{E_k\} \) if for every \( k \), \( A(E_k) \) is correct with respect to \( E_k \), and \( A(E) \) is correct with respect to \( E \).

Now we give the following incremental learning algorithm for inductive logic programs based on least general generalizations. In Section 3, we will show that this algorithm is not limit-correct in some cases.

**Algorithm \( A \):**

Input: A sequence \( \{E_k\} \) of literals such that \( E_k = E_{k-1} \cup \{l_k\} \);

Output: \( \Pi \) such that \( \Pi \vdash E_k \).

If \( k = 0 \), then return \( \Pi = \{l_0\} \);

Else if \( k = 1 \), then return \( \Pi = \{l_0, l_0 \rightarrow l_1\} \);

Else if \( A(E_{k-1}) \vdash \Pi \), then return \( \Pi = A(E_{k-1}) \);

Else if there is a literal \( l' \in A(E_{k-1}) \) such that \( (A(E_{k-1}) \setminus \{l_k\}) \vdash l' \)

then let \( A(E_{k-1}) = A(E_{k-1}) \setminus \{l'\} \) and repeat, return \( \Pi = A(E_{k-1}) \cup \{l_k\} \);

Else, \( A(E_k) = A(E_{k-1}) \cup \text{lgg}(l'' \rightarrow l', l' \rightarrow l_k) \),

where \( \rho(l', l_k) = \min_{l \in E_{k-1}} \rho(l, l_k) \),

\[ \rho(l'', l') = \min_{l \in A(E_{k-1}) \text{ and } l 
eq l''} \rho(l, l') ; \]

EndIf

Let \( \Pi = A(E_k) \);

If there is a redundant clause \( \pi' \in \Pi \) such that \( (\Pi \setminus \pi') \vdash \pi' \)

then let \( \Pi = \Pi \setminus \pi' \) and repeat;

Return \( \Pi \).

**3. The limit-correctness of algorithm \( A \)**

In this section we consider the limit-correctness of algorithm \( A \), and give two examples to show that \( A \) is not limit-correct, and sensitive to the ordering of the examples.

Let \( p \) be a predicate saying that \( x \) is an even number if \( p(x) \); and \( s \) be the successor function, i.e., \( s(x) \) is the successor of \( x \).

**Example 3.1.** Let \( E_k = \{p(0), p(s^2(0)), ..., p(s^{2k}(0))\} \).

\( A \) produces the following sequence of the Horn logic programs:

\[
A(E_0) = \{p(0)\};
A(E_1) = \{p(0); \pi_1\};
A(E_2) = \{p(0); \pi_1; \text{lgg}(\pi_1, \pi_2)\}
= \{p(0); \pi_1; p(x) \rightarrow p(s^2(x))\}
= \{p(0); p(x) \rightarrow p(s^2(x))\};
A(E_3) = A(E_2),
\]

\[
A(E_k) = A(E_2),
\]

...
where
\[
\pi_2 = \{ \neg p(s^2(0)), p(s^4(0)) \}, \\
\pi_1 = \{ \neg p(0), p(s^2(0)) \}, \\
\lgg(\pi_1, \pi_2) = \{ p(s^2(x)), \neg p(x) \}.
\]

Then, \( A(E_k) \) is a Horn logic program and the least Herbrand model of \( A(E_k) \) is
\[
M_k = \{ p(0), p(s^2(0)), ..., p(s^{2k}(0)) \}.
\]

Then we have that
\[
\Gamma = \lim_{k \to \infty} A(E_k) = A(E_1), \\
M = \lim_{k \to \infty} M_k = M_1.
\] The least Herbrand model of \( \Gamma \) is \( M \).

This example shows that \( A \) is convergent and limit-correct with respect to \( \{E_k\} \).

We change the ordering of the occurrences of \( p(s^{2k}(s))'s \) and see what \( A \) gets.

**Example 3.2.** Assume that
\[
E'_{0} = \{ p(s^4(0)) \}, \\
E'_{1} = E'_{0} \cup \{ p(s^2(0)) \}, \\
E'_{2} = E'_{1} \cup \{ p(0) \}, \\
\ldots
\]
\[
E'_{k} = E'_{k-1} \cup \{ p(s^{2k}(0)) \}, \\
\ldots
\]

Then \( A \) produces the following sequence of the Horn logic programs:
\[
A(E'_{0}) = \{ p(s^4(0)) \}; \\
A(E'_{1}) = \{ p(s^4(0)); \gamma_1 \}; \\
A(E'_{2}) = \{ p(s^4(0)); \gamma_1; \lgg(\gamma_1, \gamma_2) \} \\
= \{ p(s^4(0)); \gamma_1; p(s^2(x)) \rightarrow p(x) \} \\
= \{ p(s^4(0)); p(s^2(x)) \rightarrow p(x) \}, \\
\ldots \]
\[
A(E'_{k}) = \{ p(s^{2k}(0)); p(s^2(x)) \rightarrow p(x) \}, \\
\ldots
\]

where
\[
\gamma_1 = \{ \neg p(s^4(0)), p(s^2(0)) \}, \\
\gamma_2 = \{ \neg p(s^2(0)), p(0) \}, \\
\lgg(\gamma_1, \gamma_2) = \{ p(x), \neg p(s^2(x)) \}.
\]

Then, \( A(E'_k) \) is a Horn logic program and the least Herbrand model, say \( N_k \) of \( A(E'_k) \) is \( E'_k \). But
\[
\Pi = \lim_{k \to \infty} A(E'_k) = \{ p(s^2(x)) \rightarrow p(x) \}; \\
N = \lim_{k \to \infty} N_k = \lim_{k \to \infty} E'_k = M_1.
\]
The least Herbrand model, say \( M(\Pi) \), of \( \Pi \) is equal to the empty set. Then
\[
M(\Pi) \neq N.
\]
That is, for any \( l \in N \), \( \Pi \not\models l \).

By the above discussion, this example shows that \( A \) is not limit-correct with respect to \( \{ E_k^* \} \), and sensitive to the ordering of the examples.

4. The prioritized learning algorithm

In this section, we will first give the prioritized version of algorithm \( A \), called \( B \), which produces Horn logic programs satisfying the condition proposed in [7]. Then, we will prove that \( B \) is limit-correct to any increasing sequence of positive example sets and so not sensitive to the ordering of the examples. A clause is called simple if every subterm occurring in the body of the clause occurs in the head of the clause. A logic program is simple if every clause in it is simple.

Given a set \( E \) of positive literals, we modify algorithm \( A \) to be a new algorithm \( B \) such that \( B(E) \) is simple.

We first define a relation \( \prec \) on the literals which will be of use in Algorithm \( B \). Given two literals \( l \) and \( l' \), we say that \( l \) has a higher priority than \( l' \), denoted by \( l \prec l' \), if every sub-term occurring in \( l \) occurs in \( l' \) and \( l \neq l' \). For example, \( p(s(0)) \prec p(s^2(0)) \prec p(s^4(0)) \).

**Proposition 4.1.** \( \prec \) is a pre-order, that is, \( \prec \) is reflexive and transitive.

The prioritized algorithm \( B \) on sequences:

Input: A sequence \( \{ E_k \} \) of literals such that \( E_k = E_{k-1} \cup \{ l_k \} \);
Output: \( \Pi \) such that \( \Pi \vdash E_k \).
If there are two literals in \( E_k \) such that \( l_p \succ l_q \) where \( p < q \), then exchange the positions of these two literals and repeat;
If \( k = 0 \), then return \( \Pi = \{ l_0 \} \);
Else if \( k = 1 \), then return \( \Pi = \{ l_0, l_0 \rightarrow l_1 \} \);
Else if \( B(E_{k-1}) \vdash l_k \), then return \( \Pi = B(E_{k-1}) \);
Else if there is a literal \( l' \in B(E_{k-1}) \) such that \( (B(E_{k-1}) \cup \{ l_k \}) \vdash l' \) then let \( B(E_{k-1}) = B(E_{k-1}) \cup \{ l' \} \) and repeat, return \( \Pi = B(E_{k-1}) \cup \{ l_k \} \);
Else if \( \pi = \lgg(l'' \rightarrow l', l' \rightarrow l_k) \) is not simple, then return \( \Pi = B(E_{k-1}) \cup l_k \), where \( \rho(l', l_k) = \min_{l' \in E_{k-1}} \rho(l', l_k), \rho(l'' \rightarrow l', l' \rightarrow l_k) = \min_{l'' \in E_{k-1}} \rho(l', l'' \rightarrow l', l' \rightarrow l_k) \);
Else, \( B(E_k) = B(E_{k-1}) \cup \pi \);
EndIf
Let \( \Pi = B(E_k) \);
If there is a clause \( \pi' \in \Pi \) such that \( (\Pi \setminus \pi') \vdash \pi' \) then let \( \Pi = \Pi \setminus \pi' \) and repeat;
Return \( \Pi \).

An important step is in Algorithm \( B \) is to adjust the order of occurrences for literals in the example sets according to the pre-order \( \prec \). This can make literals with higher priority appear in the body when we derive least general generalizations for two clauses. In such a case, the probability of obtaining a simple clause will be greater. For example, \( p(x) \rightarrow p(s^2(x)) = \lgg(p(0) \rightarrow (p(s^2(0)), p(s^2(0)) \rightarrow p(s^4(0)))) \) is simple while \( p(s^2(x)) \rightarrow p(x) = \lgg(p(s^2(0)) \rightarrow p(0), p(s^4(0)) \rightarrow p(s^2(0))) \) is not.

For Algorithm \( B \), we have the following theorem.
Theorem 4.2. \( \mathcal{B} \) is convergent and limit-correct with respect to any increasing sequence \( \{E_k\} \) of literals.

Proof. We only need to prove \( \{\mathcal{B}(E_k)\} \) is convergent. Assume that when \( l_{k_0} \) is enumerated in \( E_k \) for \( k = k_0 \), a clause \( \pi \) is produced. \( \pi \) has the possibility of being taken out from \( \mathcal{B}(E_{k_1}) \) for some \( k_1 \geq k_0 \) only when \( \pi \) is deleted as a redundant clause or the example \( l_{k_1} \) enumerated in \( E_{k_1} \) has a higher priority than \( l_{k_0} \), i.e. \( l_{k_1} \prec l_{k_0} \). It is easy to see that if \( \pi \) is deleted as a redundant clause, then it will not appear in the sequence again. On the other hand, the pre-order \( \prec \), by the definition, is well-founded. That is to say, given an example \( l \), there are only finitely many \( l' \) with \( l' \prec l \). Hence, \( \pi \) cannot be put in and then be taken out from the sequence \( \{\mathcal{B}(E_k)\} \) for infinitely many times, i.e. when \( k \) is sufficiently large, either \( \pi \in \mathcal{B}(E_k) \) for every \( k \) or \( \pi \notin \mathcal{B}(E_k) \) for every \( k \). Therefore, \( \{\mathcal{B}(E_k)\} \) has the set-theoretic limit. From Algorithm \( \mathcal{B} \), we know that \( \{\mathcal{B}(E_k)\} \) is simple and so we are done.

5. Conclusion and further works

In this paper, we defined the convergence and correctness of learning algorithms with respect to infinite sequences of example sets. Then, we gave an incremental learning algorithm based on least general generalizations and showed by examples that this algorithm is not limit-correct in some cases. Finally, an algorithm, based on a priority order defined on positive literals, was proposed and it was proved that the prioritized algorithm is limit-correct with respect to any increasing infinite sequence of positive literal sets. We hope that the limit behaviors of more incremental learning algorithms will be studied in the future, which is helpful to discover useful knowledge from massive information systems.

References:


