# Learning in a Three-valued Setting

Evelina Lamma, Fabrizio Riguzzi
DEIS, Università di Bologna,
Viale Risorgimento 2
40136 Bologna, Italy,
{elamma,friguzzi}@deis.unibo.it

Luís Moniz Pereira
Centro de Inteligência Artificial (CENTRIA),
Departamento de Informática,
Universidade Nova de Lisboa,
2825 Monte da Caparica, Portugal
lmp@di.fct.unl.pt

May 8, 1998

**Abstract**

We discuss the adoption of a three-valued setting for inductive concept learning. Distinguishing between what is true, what is false and what is unknown is necessary in situations where decisions have to be taken on the basis of scarce information. We propose a learning algorithm that adopts extended logic programs under a well-founded semantics as the representation formalism and learns a definition for both the target concept and its opposite, considering positive and negative examples as instances of two disjoint classes.

In the target program, default negation is used to ensure consistency and to handle exceptions to general rules. Exceptions to a positive concept are identified from negative examples, whereas exceptions to a negative concept are identified from positive examples. Exceptions can be generalized, in their turn, resulting in a hierarchy of defaults.

## 1 Introduction

Most work on inductive concept learning considers a two-valued setting. In such a setting, what is not entailed by the learned theory is considered as false, by using the Closed World Assumption (CWA) [12]. However, in practice, it is more often the case that we know with certainty the truth or falsity of a limited number of facts and we are not able to draw any conclusion on the remaining facts, because the information available is too scarce. As it has been pointed out in [4], this is typically the case of an autonomous agent that incrementally gathers information from its surrounding world. For such an agent, it would be much better to learn in a three-valued setting and learn a definition for both the concept and its complement, using positive examples for the concept as negative examples for its complement and viceversa.

In order to represent three-valued theories of this kind, we adopt Extended Logic Programs (ELP for short) under the well-founded semantics with explicit negation *WFSX*

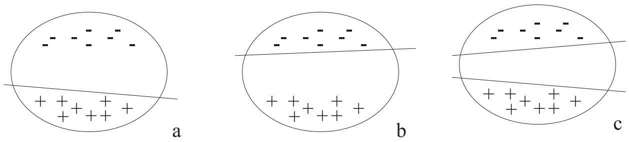


Figure 1: (a,b): two-valued setting, (c): three-valued setting (taken from [4])

[9].

We consider an extension of the Inductive Logic Programming (ILP for short) problem where the background and target theory are ELP under *WFSX*. We present an algorithm that learns a definition for both the positive concept $p$ and its (explicit) negation $\neg p$, as in [7, 4]. Coverage of examples is tested by adopting the *SLX* interpreter for ELP, defined in [2, 1]. The algorithm is also able to identify exceptions to the concepts and learn a definition for them. In turn, it can identify exceptions to exceptions and so on, thus leading to a hierachy of concepts.

The paper is organized as follows. In section 2 we introduce the new ILP framework. Section 3 presents the learning algorithm. We conclude by discussing related works in section 4.

## 2 Learning in a Three-valued Setting

In real world problems, complete information about the world is impossible to achieve and it is necessary to reason and act on the basis of the available partial information. In situations of incomplete knowledge, it is important to distinguish between what is true, what is false and what is unknown.

This is the situation of an agent that incrementally gathers information from the surrounding world and has to select its own actions on the basis of such knowledge. If the agent learns in a two-valued setting, it can encounter the problems that have been highlighted in [4]. When learning in a specific to general way, it will learn a cautious definition for the target concept and it will not be able to distinguish what is false from what is not yet known (see figure 1a). Suppose the target predicate represents the allowed actions, then the agent will not distinguish forbidden actions from actions with an unknown outcome and this can clearly be a limitation. If the agent learns in a general to specific way, instead, it will not know the difference between what is true and what is unknown (figure 1b) and therefore it can try actions with an unknown outcome. Instead, by learning in a three-valued setting, it will be able to distinguish between allowed actions, forbidden actions and actions with an unknown outcome (figure 1c). In this way, the agent will know which part of the domain needs to be further explored and will not try actions with an unknown outcome unless it is trying to expand its knowledge.

Learning in a three-valued setting requires the adoption of a more expressible class of programs to be learned. This class can be represented by means of Extended Logic Programs, under a stable semantics [6], or under a well-founded one [9]. In the following, we will adopt the well-founded semantics with explicit negation *WFSX* [9]. We will denote negation by default by *not* and explicit negation by $\neg$. $\neg A$ is said the *opposite* literal of $A$ (and viceversa) and *not* $A$ the *complementary* literal of $A$ (and viceversa).

Starting from a set of positive and negative examples for a target predicate $p$ and a background knowledge which is itself an extended logic program under *WFSX*, we apply standard ILP techniques in order to learn a definition for both the positive concept $p$ and its opposite $\neg p$. The ILP learning problem for the case of ELP has been first introduced in [7]:

**Given:**

- a set $\mathcal{P}$ of possible (extended logic) programs
- a set $E^+$ of positive examples
- a set $E^-$ of negative examples
- a consistent extended logic program $B$ (*background knowledge*)

**Find:**

- an extended logic program $P \in \mathcal{P}$ such that
  - $B \cup P \models E^+, \neg E^-$ (*completeness*)
  - $B \cup P \not\models E^-, \neg E^+$ (*consistency*)

  where $\neg E = \{\neg e | e \in E\}$, and $E^+, \neg E^-$ (resp. $E^-, \neg E^+$) stands for the conjunction of each atom in $E^+$ and $\neg E^-$ (resp. in $E^-$ and $\neg E^+$).

Note that, in the ILP problem, it is required that the program is consistent only with respect to the examples. We enlarge this condition requiring that the program is consistent also for atoms of target predicates not in the training set or *unseen atoms*. The consistency condition then becomes $B \cup P \not\models L, \neg L$ for every atom $L$ of the target concept.

Since the *SLX* procedure is correct (in the sense specified in [1]), coverage of examples is tested by adopting the *SLX* top-down interpreter for extended logic programs, defined in [1].

Our approach to learning ELP consists in applying ordinary ILP techniques to learn definitions of the positive and negative concepts. The ILP technique to be used depends on the level of generality that we want to have for the two definitions: we can look for the Least General Solution (LGS for short) or the Most General Solution (MGS for short). LGSs can be found by using a system like GOLEM [8] that adopts a bottom-up method. MGSs, instead, can be found by a top-down system, such as FOIL [11].

The generality of the solutions should be chosen independently for the two definitions, thus leading to four epistemological cases depending on the combination of solution generality for the positive and negative concept. The choice of the level of generality should be made on the basis of available knowledge on the domain. Two of the criteria that should be taken into account are the damage that can derive from an erroneous classification of an unseen object and the confidence we have in the training set.

When classifying an unseen object as belonging to a concept, we may later discover that the object belongs to the opposite concept. The more we generalize a concept, the higher number of unseen atoms is covered by the definition and the higher is the risk of an erroneous classification. Depending on the damage that may derive from such a mistake, we may decide to take a cautious or a confident approach. If the possible damage for a concept is high, then we should learn the LGS for that concept, if the possible damage is low then we can generalize the most and learn the MGS. The risk will depend on the use of the learned concepts within other rules, and so distinct generalities may be employed within the same program.

As regards the confidence in the training set, we can learn the MGS for a concept if we are confident that examples for the opposite concept are correct and representative of that concept. In fact, in top-down methods, negative examples are used in order to limit the generality of the solution. Otherwise, if we think that examples for the opposite concept are not reliable, then we should learn the LGS.

## 3 Algorithm

The algorithm that follows learns ELP of the following form:

$$\begin{array}{rcl} p(\vec{X}) & \leftarrow & p^+(\vec{X}), not\ ab_p(\vec{X}), not\ \neg p(\vec{X}) \\ \neg p(\vec{X}) & \leftarrow & p^-(\vec{X}), not\ ab_{\neg p}(\vec{X}), not\ p(\vec{X}) \end{array}$$

together with a definition for the predicates $p^+$, $p^-$ and $ab_p(\vec{X})$, $ab_{\neg p}(\vec{X})$ representing, respectively, the positive and negative concepts and exceptions to them.

$p^+$ and $p^-$ may have a non-empty intersection, thus leading to possible inconsistencies. The conflict is resolved differently depending on the type of atoms in the intersection: those that belong to one of the training sets (examples) and those that don't (unseen atoms).

Examples covered by both definitions are assigned the classification given by the training set. These atoms are considered as *exceptions* to the opposite definition and are treated by means of the non abnormality literals $not\ ab_p(\vec{X})$, $not\ ab_{\neg p}(\vec{X})$. Instead, unseen atoms are classified as unknown, since the arguments for both classifications are equally strong. This is obtained by making the rules non deterministic [1] through the addition of the literals $not\ \neg p(\vec{X})$ (resp. $not\ p(\vec{X})$).

The algorithm is given as follows:

**algorithm** LearnELP(
    **inputs :** $E^+, E^-$ : training sets,
        $B$ : background theory,
    **outputs :** $H$ : learned theory)
LearnHierarchy($E^+, E^-, B; H_p$)
LearnHierarchy($E^-, E^+, B; H_{\neg p}$)
Obtain $H$ by transforming $H_p$ and $H_{\neg p}$ into
    non-deterministic rules
**output** $H$

**procedure** LearnHierarchy(
    **inputs :** $E^+$ : positive examples,
        $E^-$ : negative examples,
        $B$ : background theory,
    **outputs :** $H$ : learned theory)
Learn($E^+, E^-, B; H_p$)
$H := H_p$
**for** each rule $r$ in $H_p$ **do**
    Find the sets $E_r^+, E_r^-$ of positive and negative

examples covered by $r$
**if** $E_r^-$ is not empty **then**
Add the literal $not\_ab_r(\vec{X})$ to $r$
Obtain $E_{ab_r}^+$ and $E_{ab_r}^-$ from $E_r^-$ and $E_r^+$ by
transforming each atom $p(\vec{t})$ into $ab_r(\vec{t})$
LearnHierarchy$(E_{ab_r}^+, E_{ab_r}^-, B; H_r)$
$H := H \cup H_r$
**endif**
**enfor**
**output** $H$

The algorithm uses procedure LearnHierarchy which, given a set of positive examples, a set of negative examples and a background knowledge, returns a definition for the positive concept, consisting of default rules, together with the definitions for the abnormality literals of a hierarchy of exceptions. The procedure LearnHierarchy is called twice, once for the positive concept and once for the negative one. In the call for the negative concept, $E^-$ is used as the positive training set and $E^+$ as the negative one.

LearnHierarchy first calls a procedure Learn$(E^+, E^-, B; H_p)$ that learns a definition $H_p$ for the target concept $p$. Learn consists of an ordinary ILP algorithm, either bottom-up or top-down, modified to adopt the $SLX$ interpreter for testing the coverage of examples and to relax the consistency requirement of the solution. The algorithm thus returns a theory that may cover some negative examples. These negative examples are then treated as exceptions, by adding a default literal to the inconsistent rules and learning a definition for the abnormality predicate. In particular, for each rule $r = p(\vec{X}) \leftarrow Body(\vec{X})$ in $H_p$, a new non-abnormality literal $not\_ab_r(\vec{X})$ is added to $r$ and a definition for $ab_r(\vec{X})$ is learned by recursively calling LearnHierarchy. Examples for $ab_r$ are obtained from examples for $p$ by observing that, in order to cover a positive (uncover a negative) example $p(\vec{t})$ for $p$, the atom $ab_r(\vec{t})$ must be false (true). Therefore, positive (negative) examples for $ab_r$ are obtained from the set $E_r^-$ of negative ($E_r^+$ of positive) examples covered by the rule.

When learning a definition for $ab_r$, in turn, LearnHierarchy may find exceptions to exceptions and call itself recursively again. In this way we are able to learn a hierarchy of exceptions.

Let us now discuss in more detail the algorithm that implements the Learn procedure. We need an algorithm that, if a consistent solution can not be found, returns a theory that covers the least number of negative examples.

Two approaches are possible. The first consists in learning the least general clause from positive examples only: since the clause is not tested on negative examples, it may cover some of them. This approach can be realized by adopting a bottom-up technique such as RLGG [10], for example by using the system GOLEM [8] that implements it, as in [7].

The second approach consists in learning from positive and negative examples adopting a top-down learning algorithm where consistency of clauses (specialization loop stopping criterion in top-down algorithms) is replaced by a weaker requirement. The simplest criterion that can be adopted is to stop specializing the clause when no literal can be added that reduces the coverage of negative examples.

## 4 Related Work

The problems raised by negation and uncertainty in concept-learning, and Inductive Logic Programming in particular, were pointed out in some previous work (e.g., [3, 4]). For concept learning, the use of the CWA for target predicates is no longer acceptable because it does not allow to distinguish between what is false and what is undefined. To avoid this problem, De Raedt and Bruynooghe [4] proposed to use a three valued logic and an explicit definition of the negated concept in concept learning. This technique has been integrated within the CLINT system, an interactive concept-learner. In the resulting system, both a positive and a negative definition are learned for a concept (predicate) $p$, stating, respectively, the conditions under which $p$ is true and false. Furthermore, it is required that the concept descriptions be consistent.

The system LELP (Learning ELP) [7] learns ELP under answer-set semantics. As our algorithm, LELP is able to learn non-deterministic default rules with a hierarchy of exceptions. From the point of view of the learning problems that the two algorithms can solve, they are equivalent when the background is a definite logic program: all the examples shown in [7] can be learned by our algorithm.

When the background is an ELP, instead, the adoption of a well-founded semantics gives a number of advantages with respect to the answer-set semantics. For non-stratified background theories, answer-sets semantics does not enjoy the structural property of relevance [1], like our *WFSX* does, and so they cannot employ top-down proof procedures. For the well-founded semantics, instead, the top-down *SLX* interpreter is available, that can be used for testing the coverage of examples in the learning algorithm.

A difference between us and [7] is in the level of generality of the definitions they can learn. LELP generate clauses from positive examples only therefore it can only employ a bottom-up ILP technique and learn the LGS. Instead, we can choose whether to adopt a bottom-up or a top-down algorithm and we can learn theories of different generality for different target concepts.

Several other authors have also addressed the task of learning rules with exceptions [5, 13]. In these frameworks, nonmonotonicity and exceptions are dealt with by learning logic programs with negation. In [5] the authors rely on a language which uses a limited form of "classical" (or, better, syntactic) negation together with a priority relation among the sentences of the program which can be easily mapped into negation as default.

## Acknowledgments

This research was partially funded by the PRAXIS XXI project MENTAL, and a NATO sabbatical scholarship to L. M. Pereira.

## References

[1] J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*, volume 1111 of *LNAI*. Springer-Verlag, Heidelberg, 1996.

[2] J.J. Alferes, C.V. Damásio, and L.M. Pereira. Top-down query evaluation for well-founded semantics with explicit negation. In *Proceedings of the European Conference on Artificial Intelligenece ECAI94*, pages 140–144. Morgan Kaufmann, 1994.

[3] M. Bain and S. Muggleton. Non-monotonic learning. In S. Muggleton, editor, *Inductive Logic Programming*, pages 145–161. Academic Press, 1992.

[4] L. De Raedt and M. Bruynooghe. On negation and three-valued logic in interactive concept learning. In *Proceedings of the 9th European Conference on Artificial Intelligence*, 1990.

[5] Y. Dimopoulos and A. Kakas. Learning Non-monotonic Logic Programs: Learning Exceptions. In *Proceedings of the 8th European Conference on Machine Learning*, 1995.

[6] M. Gelfond and V. Lifschitz. Classical negation in logic programming and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[7] Katsumi Inoue and Yoshimitsu Kudoh. Learning extended logic programs. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 176–181. Morgan Kaufmann, 1997.

[8] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan, 1990.

[9] L.M. Pereira and J.J. Alferes. Well founded semantics for logic programs with explicit negation. In *Proceedings of the European Conference on Artificial Intelligenece ECAI92*, pages 102–106. John Wiley and Sons, 1992.

[10] G.D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[11] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[12] R. Reiter. On closed-word data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, 1978.

[13] A. Srinivasan, S. Muggleton, and M. Bain. Distinguishing exceptions from noise in non-monotonic learning. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.