Learning Three-Valued Logic Programs

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),
Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa,
2825-114 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract

We show that the adoption of a three-valued setting for inductive concept learning is particularly useful for learning. Distinguishing between what is true, what is false and what is unknown can be useful in situations where decisions have to be taken on the basis of scarce information.

In order to learn in a three-valued setting, we adopt Extended Logic Programs (ELP) under a Well-Founded Semantics with explicit negation (*WFSX*) as the representation formalism for learning. Standard Inductive Logic Programming techniques are then employed to learn the concept and its opposite.

The learnt definitions of the positive and negative concepts may overlap. In the paper, we handle the issue of combination of possibly contradictory learnt definitions, and we show strategies for theory refinement.

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 false, on the basis of the Closed World Assumption (CWA) [21]. However, in practice, it is more often the case that we are confident about the truth or falsity of only a limited number of facts, and are not able to draw any conclusion about the remaining ones, because the available information is too scarce. Like it has been pointed out in [5, 4], this is typically the case of an autonomous agent that, in an incremental way, gathers information from its surrounding world. Such an agent needs to distinguish between what is true, what is false and what is unknown, and therefore needs to learn within a richer three-valued setting.

For this purpose, we adopt the class of *Extended Logic Programs* (ELP, for short, in the sequel) as the representation language for learning in a three-valued setting. ELP contain two kinds of negation: default negation plus a second form of negation, called *explicit*, whose combination has been recognized as a very expressive means of knowledge representation. The adoption of ELP allows one to deal directly in the language with incomplete knowledge, with exceptions through default negation, as well as with truly *negative* information through explicit negation [17, 1, 2]. For instance, in [1, 3, 12] it is shown how ELP are applicable to such diverse domains of knowledge representation as concept hierarchies, reasoning about actions, belief revision, counterfactuals, diagnosis, updates and debugging.

In this work, we first discuss various approaches and strategies that can be adopted in Inductive Logic Programming (ILP, henceforth) for learning with a three-valued settings.

As in [7], the learning process starts from a set of positive and negative examples plus some background knowledge in the form of an extended logic program. Positive and negative information in the training set are treated equally, by learning a definition for both a positive concept p and its (explicitly) negated concept $\neg p$ by means of standard ILP techniques. Coverage of examples is tested by adopting the *S*LX interpreter for ELP under the Well-Founded Semantics with explicit negation (*WFSX*) defined in [1].

Indeed, separately learned positive and negative concepts may conflict and, in order to handle possible *contradiction*, contradictory learned rules are defused by making the learned definition for a positive concept p depend on the default negation of the negative concept $\neg p$, and vice-versa. I.e., each definition is introduced as an exception to the other. Moreover, the intersection of the solutions learned can provide useful hints for theory refinement.

2 Extended Logic Programs

An *extended logic program* is a finite set of rules of the form:

$$L_0 \leftarrow L_1, \ldots, L_n$$

with $n \geq 0$, where L_0 is an objective literal, $L_1, \ldots, L_n$ are literals and each rule stands for the sets of its ground instances. *Objective literals* are of the form A or $\neg A$, where A is an atom, while a *literal* is either an objective literal L or its default negation *not* L. $\neg A$ is said the *opposite* literal of A (and vice versa), where $\neg\neg A = A$, and *not* A the *complementary* literal of A (and vice versa). By *not* $\{a_1, \ldots, a_n\}$ we mean $\{not\ a_1, \ldots, not\ a_n\}$ where a_is are literals. By $\neg\ \{a_1, \ldots, a_n\}$ we mean $\{\neg a_1, \ldots, \neg a_n\}$. The set of all objective literals of a program P is called its *extended Herbrand base* and is represented as $H^E(P)$. An *interpretation* I of an extended program P is denoted by $T \cup not\ F$, where T and F are disjoint subsets of $H^E(P)$. Objective literals in T are said to be *true* in I, objective literals in F are said to be *false* in I and those in $H^E(P) - I$ are said to be *undefined* in I. We introduce in the language the proposition $\mathbf{u}$ that is undefined in every interpretation I.

The *WFSX* extends the well founded semantics (*WFS*) [22] for normal logic programs to the case of extended logic programs. *WFSX* is obtained from *WFS* by adding the coherence principle relating the two forms of negation: "*if L is an objective literal and $\neg L$ belongs to the model of a program, then also not L belongs to the model*", i.e., $\neg L \rightarrow not\ L$. See [1, 6] for a definition of *WFSX*.

Let us now show an example of the *WFSX* semantics in the case of a simple program.

Example 1 *Consider the following extended logic program:*

$\neg a \leftarrow .$ $\qquad\qquad$ $b \leftarrow not\ b.$
$a \leftarrow b.$

A WFSX *model of this program is $M = \{\neg a, not\ \neg b, not\ a\}$: $\neg a$ is true, a is false (i.e., both $\neg a$ and not a are in the well-founded model), $\neg b$ is false (there are no rules for $\neg b$) and b is undefined. Notice that not a is in the model since it is implied by $\neg a$ via the coherence principle.*

WFSX provides a semantics for the class of extended logic programs: the set of *non-stratified* programs, i.e., the set of programs that contain recursion through default negation.

Non-stratified programs are very useful for knowledge representation because the *WFSX* semantics assigns the truth value undefined to the literals involved in the recursive cycle through negation. In section 4 we will employ non stratified programs in order to resolve possible contradictions.

For *WFSX* there exists a top-down proof procedure *SLX* [1], which is correct with respect to the semantics.

3 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 or undefined.

Learning in a three-valued setting requires the adoption of a more expressive class of programs to be learned. This class can be represented, we have seen, by means of extended logic programs under the well-founded semantics extended with explicit negation *WFSX* [1, 2, 17].

We therefore consider a new learning problem where we want to learn an extended logic program from a background knowledge that is itself an extended logic program and from a set of positive and a set of negative examples in the form of ground facts for the target predicates.

Our approach to learning with extended logic programs consists in initially applying conventional ILP techniques to learn a positive definition from E^+ and E^- and a negative definition from E^- and E^+. In these techniques, the *SLX* procedure substitutes the standard Logic Programming proof procedure to test the coverage of examples (see [9]).

The ILP techniques to be used depend on the level of generality that we want to have for the two definitions: we can look for the Least General Solution (LGS) or the Most General Solution (MGS) of the problem of learning each concept and its complement. In practice, LGS and MGS are not unique and real systems usually learn theories that are not the least nor most general, but approximate one of the two. In the following, these concepts will be used to signify approximations of the theoretical concepts.

LGSs can be found by adopting one of the bottom-up methods such as relative least general generalization (*rlgg*) [18] and the GOLEM system [16], inverse resolution [15] or inverse entailment [10]. Conversely, MGSs can be found by adopting a top-down refining method (cf. [11]) and a system such as FOIL [20] or Progol [14].

A system has been implemented that is able to solve the above mentioned problem. The system is called LIVE (Learning in a three-Valued Environment) and is described in detail in [9]. In particular, the system learns

a definition for both the concept and its opposite and is parametric in the procedure used for learning each definition: it can adopt either a top-down algorithm, using beam-search and heuristic necessity stopping criterion, or a bottom-up algorithm, that exploits the GOLEM system. The code of the overall system can be downloaded from the site: `http://www-lia.deis.unibo.it/Software/LIVE/`.

4 Strategies for Eliminating Learned Contradictions

The definitions learned for the positive and negative concepts may overlap. In this case, we have a contradictory classification for the objective literals in the intersection. In order to resolve the conflict, we must distinguish two types of literals in the intersection: those that belong to the training set and those that do not, also dubbed *unseen* atoms (see figure 1).

In the following, we discuss how to resolve the conflict in the case of unseen literals and of literals in the training set. From now onwards, $\vec{X}$ stands for a tuple of arguments.

Contradiction on Unseen Literals For unseen literals, the conflict is resolved by classifying them as undefined, since the arguments supporting the two classifications are equally strong. Instead, for literals in the training set, the conflict is resolved by giving priority to the classification stipulated by the training set. In other words, literals in a training set that are covered by the opposite definition are made as *exceptions* to that definition. For unseen literals in the intersection, the undefined classification is obtained by making opposite rules mutually

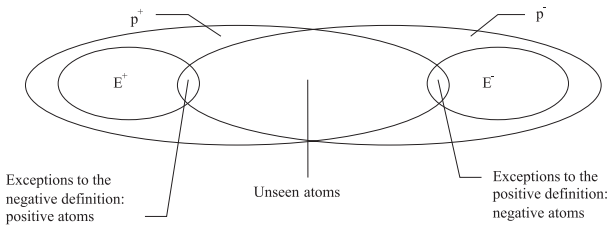

Figure 1: Interaction of the positive and negative definitions on exceptions.

defeasible, or "non-deterministic" (see [3, 1]). The target theory is consequently expressed in the following way:

$$\begin{aligned} p(\vec{X}) &\leftarrow p^+(\vec{X}), not\ \neg p(\vec{X}) \\ \neg p(\vec{X}) &\leftarrow p^-(\vec{X}), not\ p(\vec{X}) \end{aligned}$$

where $p^+(\vec{X})$ and $p^-(\vec{X})$ are, respectively, the definitions learned for the positive and the negative concept, obtained by renaming the positive predicate by p^+ and its explicit negation by p^-. From now onwards, we will indicate with these superscripts the definitions learned separately for the positive and negative concepts.

We want $p(\vec{X})$ and $\neg p(\vec{X})$ each to act as an exception to the other. In case of contradiction, this will introduce mutual circularity, and hence undefinedness according to *WFSX*. For each literal in the intersection of p^+ and p^-. According to *WFSX*, there is a (partial) stable model where no literal $p(\vec{X})$, $\neg p(\vec{X})$, $not\ p(\vec{X})$ or $not\ \neg p(\vec{X})$ belongs to the model. This is the least partial stable model and represents the well-founded model of the theory.

Contradiction on Examples Theories are tested for consistency on all the literals of the training set, so we should not have a conflict on them. However, in some cases, it is useful to relax the consistency requirement and learn clauses that cover a small amount of counter examples. This is advantageous when it would be otherwise impossible to learn a definition for the concept, because no clause is contained in the language bias that is consistent, or when an overspecific definition would be learned, composed of very many specific clauses instead of a few general ones. In such cases, the definitions of the positive and negative concepts may cover examples of the opposite training set. These must then be considered exceptions and treated as abnormalities.

Exceptions are due to abnormalities in the opposite concept. To handle exceptions to classification rules, we add a negative default literal of the form $not\ abnorm_p(\vec{X})$ (resp. $not\ abnorm_{\neg p}(\vec{X})$) to the rule for $p(\vec{X})$ (resp. $\neg p(\vec{X})$), to express possible abnormalities arising from exceptions. Then, for every exception $p(\vec{t})$, an individual fact of the form $abnorm_p(\vec{t})$ (resp. $abnorm_{\neg p}(\vec{t})$) is asserted so that the rule for $p(\vec{X})$ (resp. $\neg p(\vec{X})$) does not cover the exception, while the opposite definition still covers it. In this way, exceptions will figure in the model of

the theory with the correct truth value. The learned theory thus takes the form:

$$p(\vec{X}) \leftarrow p^{+}(\vec{X}), not\ abnorm_{p}(\vec{X}), not\ \neg p(\vec{X}) \quad (1)$$

$$\neg p(\vec{X}) \leftarrow p^{-}(\vec{X}), not\ abnorm_{\neg p}(\vec{X}), not\ p(\vec{X}) \quad (2)$$

Individual facts of the form $abnorm_{p}(\vec{X})$ are then used as examples for learning a definition for $abnorm_{p}$ and $abnorm_{\neg p}$, as in [7, 8]. In turn, exceptions to the definitions of $abnorm_{p}$ and $abnorm_{\neg p}$ may be found and so on, thus leading to a hierarchy of exceptions.

These techniques have been implemented in the system LIVE: the system learns a definition for a concept and its opposite and then combines them by means of non-deterministic rules. Moreover, the system is able to identify exceptions and to learn a hierarchical definition for them.

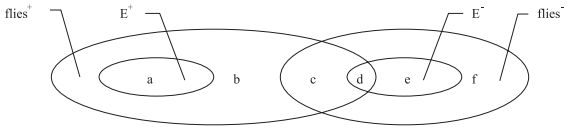

Figure 2: Coverage of definitions for opposite concepts

The example above and figure 2 show all the various cases for a literal when learning in a three-valued setting. a and e are examples that are consistently covered by the definitions. b and f are unseen constants on which there is no contradiction. c and d are constants where there is contradiction, but c is classified as undefined whereas d is considered as an exception to the positive definition and is classified as negative.

5 Strategies for Theory Refinement

As we have shown, when learning a definition for a concept p and its opposite $\neg p$ (separately or not), it can be the case that some contradiction arises for an *unseen* literal. Figure 3 depicts various cases which may occur. Identifying such contradictions is useful in interactive theory revision, where the system can ask an oracle to classify the literal(s) leading to contradiction, and accordingly revise the least general solutions (LGS) or most general solutions (MGS) for p and for $\neg p$. Detecting uncovered literals thus points to theory extension.

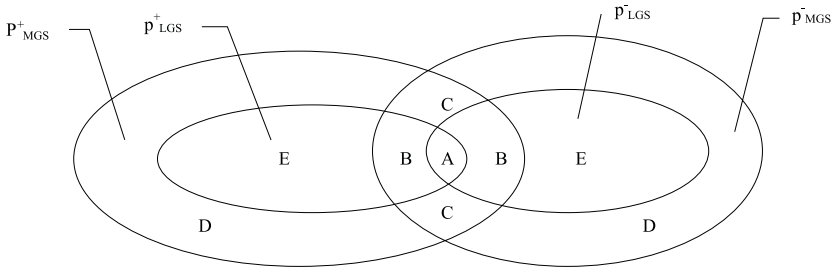

Figure 3: Intersection of Learnt Solutions

Further information on unseen contradictory literals for the various cases can help in improving learnt rules.

Area A represents contradictions between the two least general solutions, for a concept p and its opposite $\neg p$, i.e., it represents unseen literals satisfying the conjunction $p^{+}_{LGS}(\vec{X}), p^{-}_{LGS}(\vec{X})$. This is the strongest form of contradiction, and unseen literals in region A should be given priority when querying the oracle.

Areas B represent contradictions between most general solutions for concept p^{+} and p^{-} which are outside the least general solution for one concept, but inside the least general solution for the other. I.e., they represent unseen literals satisfying the conjunction $p^{+}_{MGS}(\vec{X}), p^{-}_{LGS}(\vec{X}), not\ p^{+}_{LGS}(\vec{X})$ or the conjunction $p^{-}_{MGS}(\vec{X}), p^{+}_{LGS}(\vec{X})$, $not\ p^{-}_{LGS}(\vec{X})$. For literals satisfying the first conjunction, the system has to revise most general solution for p^{+} if the oracle classifies the literal as negative and the least and most general solution for p^{-} if the oracle classifies the literal as positive; vice-versa for the literals satisfying the second conjunction.

Though less strongly contradictory than area A, areas B are more strongly so than areas C, and so merit attention next when querying the oracle.

Areas C represent contradictions between most general solutions for concept p and its opposite which are outside both the least general solutions. I.e., it represents literals satisfying the conjunction $p^+_{MGS}(\vec{X}), not\ p^+_{LGS}(\vec{X})$, $p^-_{MGS}(\vec{X}), not\ p^-_{LGS}(\vec{X})$. Identifying such contradictions can be useful in refining knowledge and bridging the gap between most and least general solution for a concept. The system has to revise the most general solution for p^+ if the oracle classifies the atom as negative and for p^- if the oracle classifies the atom as positive, and vice-versa.

Finally, it is worth mentioning that other regions where a contradiction does not arise, namely D and E, can be useful in guiding knowledge acquisition. New information about an unseen atom always increases knowledge, and thus eventually requires knowledge refinement or knowledge extension. However, among unseen literals not leading to contradiction, we can identify class D which can be more useful than E in bridging the gap between least and most general solution. This area represents instances which satisfy the conjunction $p^+_{MGS}(\vec{X}), not\ p^+_{LGS}(\vec{X}), not\ p^-_{MGS}(\vec{X})$ or the conjunction $p^-_{MGS}(\vec{X}), not\ p^-_{LGS}(\vec{X}), not\ p^+_{MGS}(\vec{X})$. If a literal satisfying the former condition is classified as negative by an oracle, then the most general solution for p^+ has to be revised, whereas if a literal satisfying the latter condition is classified as positive by an oracle, then the most general solution for p has to be revised.

It may be that learnt rules do not cover atoms and their negations for legitimate argument tuples. Accordingly, a further area exists (the one outside the areas in figure 3) which pinpoints cases of interest, leading to theory extension (and subsequent refinement where contradictions emerge).

6 Related Work

The adoption of negation in learning has been investigated by many authors. Many propositional learning systems learn a definition for both the concept and its opposite. For example, systems that learn decision trees, as c4.5 [19], or decision rules, as the AQ family of systems [13], are able to solve the problem of learning a definition for n classes, that generalizes the problem of learning a concept and its opposite. However, in most cases the definitions learned are assumed to cover the whole universe of discourse: no undefined classification is produced, any instance is always classified as belonging to one of the classes. Instead, we classify as undefined the instances for which the learned definitions do not give an unanimous response.

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 [5, 4]: De Raedt and Bruynooghe [5] proposed to use a three-valued logic (later on formally defined in [4]) 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 those under which it is false. The definitions are learned so that they do not produce an inconsistency on the examples. In order to take care of consistency on unseen examples, CLINT asserts an integrity constraint $pandnot\_p \rightarrow false$ and takes care that the constraint is never violated. Differently from this system, we are able to learn definitions for exceptions to both concepts. Furthermore, we are able to cope with two kinds of negation, the explicit one used to state what is false, and the default (defeasible) one used to state what can be assumed false.

The system LELP (Learning Extended Logic Programs) [7] also learns extended logic programs. We differ from the system LELP [7] in two respects: first, we always learn a definition for a concept and its opposite from examples, while LELP does so only when there is a near even number of positive and negative examples, second, we adopt a well-founded semantics.

7 Conclusions

The two-valued setting that has been considered in most work on ILP and Inductive Concept Learning in general is not sufficient in many cases where we need to represent real world data. This is for example the case of an agent that has to learn the effect of the actions it can perform on the domain by performing experiments. Such an agent needs to learn a definition for allowed actions, forbidden actions and actions with an unknown outcome and therefore it needs to learn in a richer three-valued setting.

In order to adopt such a setting in ILP, the class of extended logic programs under the well-founded semantics with explicit negation ($WFSX$) is adopted as the representation language. This language allows two kinds of negation, default negation plus a second form of negation called explicit, that is used in order to represent explicitly negative information. Adopting extended logic programs in ILP prosecutes the general trend in Machine Learning of extending the representation language in order to overcome the limits of existing systems.

Contradictions may arise in the programs that are learned. We consider two cases: in the first the contradiction arises for unseen literals and is solved via non-deterministic rules, while in the second the contradiction arises on

examples and is solved via default literals added as non-abnormality conditions to rules.

References

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

[2] J. J. Alferes, L. M. Pereira, and T. C. Przymusinski. "Classical" negation in non-monotonic reasoning and logic programming. *Journal of Automated Reasoning*, 20:107–142, 1998.

[3] C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

[4] L. De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press, 1992.

[5] 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.

[6] J. Dix, L. M. Pereira, and T. Przymusinski. Prolegomena to logic programming and non-monotonic reasoning. In J. Dix, L. M. Pereira, and T. Przymusinski, editors, *Non-Monotonic Extensions of Logic Programming - Selected papers from NMELP'96*, number 1216 in LNAI, pages 1–36, Germany, 1997. Springer-Verlag.

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

[8] E. Lamma, P. Mello, M. Milano, and F. Riguzzi. Integrating induction and abduction in logic programming. In P. P. Wang, editor, *Proceedings of the Third Joint Conference on Information Sciences*, volume 2, pages 203–206, 1997.

[9] E. Lamma, F. Riguzzi, and L. M. Pereira. Strategies in combined learning via logic programs. to appear in Machine Learning, 1999.

[10] S. Lapointe and S. Matwin. Sub-unification: A tool for efficient induction of recursive programs. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 273–281. Morgan Kaufmann, 1992.

[11] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[12] J. A. Leite and L. M. Pereira. Generalizing updates: from models to programs. In J. Dix, L. M. Pereira, and T. C. Przymusinski, editors, *Collected Papers from Workshop on Logic Programming and Knowledge Representation LPKR'97*, number 1471 in LNAI. Springer-Verlag, 1998.

[13] R. Michalski. Discovery classification rules using variable-valued logic system VL1. In *Proceedings of the Third International Conference on Artificial Intelligence*, pages 162–172. Stanford University, 1973.

[14] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[15] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In S. Muggleton, editor, *Inductive Logic Programming*, pages 261–280. Academic Press, 1992.

[16] 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.

[17] 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.

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

[19] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

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

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

[22] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.