Learning with Extended Logic Programs

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

Paola Mello
Dip. di Ingegneria, Università di Ferrara,
Via Saragat 1, 44100 Ferrara, Italy
pmello@ing.unife.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

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 can be useful in situations where decisions have to be taken on the basis of scarce information. In a three-valued setting, we want to learn a definition for both the target concept and its opposite, considering positive and negative examples as instances of two disjoint classes. To this purpose, we adopt extended logic programs under a well-founded semantics as the representation formalism for learning. In this way, we are able to represent both the concept and its opposite and deal with incomplete or unknown information.

We discuss various approaches to be adopted in order to handle possible inconsistencies. 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, by learning within 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) [20]. 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 [6], this is typically the case of an autonomous agent that gathers information incrementally from its surrounding world. The agent has to choose its actions on the basis of the knowledge it possesses and knowing that an action certainly leads to a failure is different from not knowing anything about its outcome. It will never try an action when it is sure of its negative effect, but it may try an action with an unknown outcome in situations where no other action can be taken or in order to expand its knowledge.

Therefore, for such an agent, it would be much better to be able to distinguish between what is certainly true, what is certainly false and what is unknown. To this purpose, the agent should adopt 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. The learned theory will then classify instances in three ways: instances covered by the positive definition are positive, instances covered by the negative definition are negative and instances not covered by any definition are unknown.

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* [17]. In [19, 1] it is shown how ELP can be applied to domains where negative information is made symmetric to positive one, e.g. concept hierarchies, reasoning about actions, counterfactuals, diagnosis, and debugging.

In this work, we consider an extension of Inductive Logic Programming (ILP for short) in order to learn ELP under *WFSX*. As in [14, 6], we learn a definition for both a positive concept p and its (explicit) negation $\neg p$. Coverage of examples is tested by adopting the *SLX* interpreter for ELP, defined in [2, 1].

When learning both positive and negative concepts, we may have inter-

action between the two: their definitions may have a non-empty intersection. We must distinguish two types of atoms in the intersection. *Unseen* atoms, i.e., atoms not present in the training set, should be assigned an unknown value, while atoms in the training set should be assigned the truth value of the training set to which they belong and be considered as *exceptions* to the opposite definition. Therefore, exceptions to a positive concept are identified from negative examples, whereas exceptions to a negative concept are identified from positive examples.

Explicit negation is used to represent the opposite concept while default negation is used to ensure consistency and to handle exceptions to general rules. Exceptions can be generalized, in their turn, by learning within a hierarchy of defaults.

Major innovations of the work concern both the application of ILP to the case of ELP, by integrating standard ILP algorithms with a top-down interpreter for ELP under a well-founded semantics and the discussion of various approaches to be adopted in order to deal with consistency and exceptions.

The paper is organized as follows: in Section 2 we further motivate the need for learning ELP under well-founded semantics, and formally introduce a new ILP framework. Section 3 discusses how to avoid inconsistencies through the use of non-deterministic rules and a meta-level $undefined$ predicate. This section also shows how to deal with exceptions through negation as default. Related works are discussed in Section 4. Section 5 concludes and presents directions for future works.

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, so that decision can be made exploiting the available information as much as possible.

Therefore, an agent acquiring knowledge about the world should perform learning in a three-valued settings. This requires the adoption of a more expressible class of programs to be learned, where some atoms are true, some false and some others are unknown or undefined. This class can be

represented by means of Extended Logic Programs, under a stable semantics [12], or under a well-founded one [17]. In the following, we will adopt the well-founded semantics with explicit negation *WFSX* [17]. 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 [14] and is given as follows.
Given:

- a set $\mathcal{P}$ of possible (extended logic) programs
- a set E^+ of positive examples
- a set E^- of negative examples
- an 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 ILP problem, it is required that the program is consistent only with respect to the examples. This requirement can be enlarged to require that the program is consistent in the general sense, i.e. $B \cup P \not\models L, \neg L$ for every literal L.

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 [2, 1]. It is important to note that neither answer-sets, nor three-valued strong negation, enjoy relevance [1, 3], i.e., they cannot

have top-down querying procedures, and that is why we use SLX, for $WFSX$ is relevant. We say that an example e is *covered* by program P if $P \vdash e$ according to the SLX procedure.

Therefore the conditions that a program P must satisfy in order to be a solution to the ILP problem can be expressed as "P must cover all the positive examples and all explicit negations of the negative ones". A theory that satisfies the previous condition is said to be *complete*, while a theory that does not cover both some literal and its explicit negation is said to be *consistent*.

The set $\mathcal{P}$ is called the *hypothesis space*. The importance of this set lies in the fact that it defines the search space of the ILP system. In order to be able to effectively learn a program, this space must be restricted as much as possible. If the space is too big, the search could result infeasible.

3 Intersection of Positive and Negative Definitions

The definitions of the positive and negative concept may overlap. In this case, we have a contradictory classification for the atoms in the intersection. In order to resolve the conflict, we must distinguish two types of atoms in the intersection: those that belong to the training set and those that don't, also called *unseen* atoms (see figure 1).

For unseen atoms, the conflict should be resolved by classifying the atoms as unknown, since the arguments for both classifications are equally strong. Instead, for atoms in the training set, the conflict should be resolved by giving priority to the classification given by the training set, supposing this information is reliable, i.e., in the hypothesis of absence of noise. In other words, atoms in the training set that are covered by the opposite definition should be considered as *exceptions* of that definition.

For unseen atoms in the intersection, the unknown classification is obtained by making the rules non deterministic (see [18, 5, 1]). The target theory is thus expressed in the following way:

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

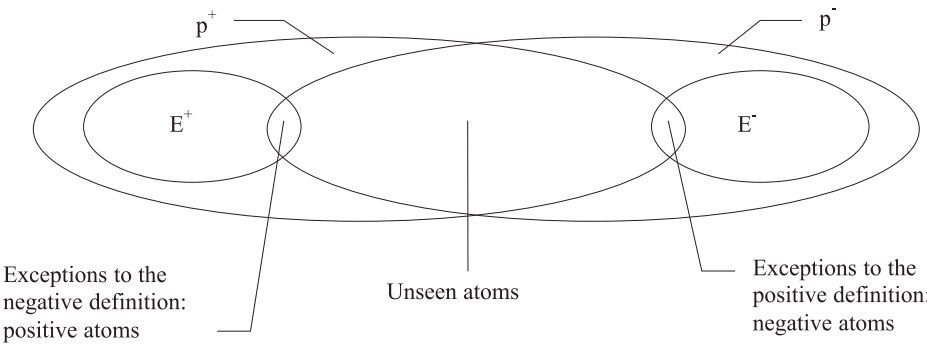

Figure 1: Interaction of the positive and negative definitions on exceptions.

where $p^+(\vec{X})$ and $p^-(\vec{X})$ are, respectively, the definitions learned for the positive and the negative concept. For each atom in the intersection, there are two partial stable models, one containing the atom in its positive version, the other containing the opposite literal. The atom is unknown, according to the well-founded semantics for explicit negation [17], i.e in the least partial stable model.

Note that the program $B \cup P$ can be non-stratified programs, either because the original background is already non-stratified or because the learned program is non-stratified. In this case, three-valued semantics can produce literals with the value "unknown" and one or both of p^+ and p^- may be unknown. If one is unknown and the other is true, then the rules above make both p and $\neg p$ undefined, since the negation by default of an undefined atom is undefined. However, this is counter-intuitive: the defined value should prevail.

In order to handle this case, we suppose to have available a system predicate $undefined(X)$ that succeeds iff the atom X is undefined. So we add the following two rules to the definitions for p and $\neg p$:

$$\begin{aligned} p(\vec{X}) &\leftarrow p^+(\vec{X}), undefined(\neg p(\vec{X})) \\ \neg p(\vec{X}) &\leftarrow p^-(\vec{X}), undefined(p(\vec{X})) \end{aligned}$$

According to these clauses, p is true when p^+ is true and $\neg p$ is undefined.

Let us consider now the case in which some atoms in the intersection belong to the opposite training set. We want to classify these atoms according

to the classification given by the training set. To this purpose, we add a negative literal (using negation as default) of the kind $not\ abnorm_{p^+}(\vec{X})$ ($not\ abnorm_{p^-}(\vec{X})$) to the rule for p ($\neg p$), expressing the default condition. Then, for every exception $p(\vec{t})$, an individual fact of the form $abnorm_{p^+}(\vec{t})$ ($abnorm_{p^-}(\vec{t})$) is asserted (and possibly generalized) so that the rule for p ($\neg p$) will not cover the exception. In this way, exceptions will be present in the model of the theory with the correct definition. The rules thus take the following form:

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

Abnormality literals have not been added to the rules for the undefined case because an atom that is an example can not be undefined, since it must be covered by the respective definition.

In this way, we are able to deal both with the case in which the two definitions are inconsistent and with the case in which exceptions to rules exist, as it is shown in the next example.

Example 1 *Consider a domain containing entities a, b, c, d, e, f and suppose the target concept is fly. Let the background knowledge be:*

$bird(a).$	$has_wings(a).$	
$jet(b).$	$has_wings(c).$	
$quetzalcoatl(c)$	$has_wings(c).$	$has_limbs(c).$
$penguin(d)$	$has_wings(d).$	$has_limbs(d).$
$dog(e).$	$has_limbs(e).$	
$cat(f).$	$has_limbs(f).$	

and let the training set be:

$E^+ = \{fly(a)\}$
$E^- = \{fly(d), fly(e)\}$

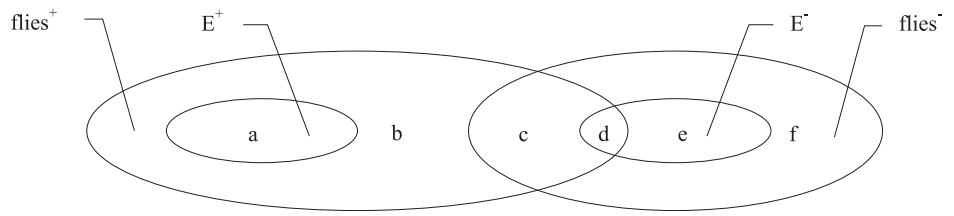

Figure 2: Coverage of definitions for the positive and negative concept

The learned theory is:

$$
\begin{array}{rcl}
flies(X) & \leftarrow & flies^{+}(X), not\ abnorm_{flies^{+}}(X), not\ \neg flies(X). \\
\neg flies(X) & \leftarrow & flies^{-}(X), not\ abnorm_{flies^{-}}(X), not\ flies(X). \\
flies(X) & \leftarrow & flies^{+}(X), undefined(\neg flies(X)). \\
\neg flies(X) & \leftarrow & flies^{-}(X), undefined(flies(X)). \\
abnorm_{flies^{+}}(d) & \leftarrow & .
\end{array}
$$

where

$$
\begin{array}{rcl}
flies^{+}(X) & \leftarrow & has_wings(X). \\
flies^{-}(X) & \leftarrow & has_limbs(X).
\end{array}
$$

This example shows all the various cases for an atom when learning in a three-valued setting. a and e are examples that are consistently covered by the definitions. b and f are unseen atoms on which there is no contradiction. c and d are atoms where there is contradiction, but c is classified as unknown whereas d is considered as an exception to the positive definition and is classified as negative.

The probability and type of interaction depend on the level of generality of the definitions learned for p^{+} and p^{-}. The higher is the generality, the higher is the probability of interaction between the two. Different levels of generality can be obtained by using different algorithms: Least General Solutions (LGS for short) can be obtained by bottom-up algorithms, and Most General Solutions (MGS for short) by top-down algorithms.

The generality of the solutions can be chosen independently for the two definitions, thus leading to four interaction types.

It is worth mentioning that the level of generality can be related with the *risk of damage* that can derive from an erroneous classification of an unseen object. We can learn MGS for both p^+ and p^- when the risk of damage is low for both the positive and negative concept; LGS for both p^+ and p^- when the risk is high for both the positive and negative concept; MGS for p^+ and LGS for p^- when the risk of damage is low for the positive concept and high for the negative one; LGS for p^+ and MGS for p^- when the risk of damage is high for the positive concept and low for the negative one. The risk and generality will depend on the use of the learned concepts within other rules, and so distinct generalities may be employed within the same program.

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., [4, 6]). For negated concept learning, negation as failure (and the underlying use of CWA) is no longer acceptable (except for coding exceptions or abnormalities), since it acts as if everything is known. As pointed out in [6], this leads to a paradox since if everything is known then nothing ought to be learned. Furthermore, negation increases the search space for a concept, since the expressivity of the concept description language grows, and allowing negation often amounts to allowing disjunctions in concept learning. To avoid these problems, De Raedt and Bruynooghe [6] 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.

In a recent work, Inoue and Kudoh [14] present a method for learning ELP in Inductive Logic Programming, under answer-sets semantics. In particular, their system (called LELP, Learning ELP) learns default rules with exceptions in the form of ELP given incomplete positive and negative examples and background knowledge. LELP first generates candidate rules from positive examples only (resp. negative examples only) and background knowledge in an ordinary ILP (bottom-up) framework. Both concepts are

learned only when the number of positive examples is close to that of negative ones. Then exceptions are identified (as covered examples of the opposite set) and rules specialized through negation as default and abnormality atoms, which are then assumed to prevent the coverage of exceptions. These assumptions can be, in their turn, generalized to generate hierarchical default rules. Finally, in order to generate consistent programs, learned rules are made non-deterministic in the context of answer set semantics, by means of non-stratified rules. We rely, instead, on a well-founded semantics for ELP, and the SLX interpreter in particular, which is computationally less expensive than the answer set semantics, and allows for top-down query procedures. For non-stratified programs, which the authors require, answer-sets semantics does not enjoy the structural property of relevance [8, 9], like our WFSX does, and so they cannot employ top-down querying methods, nor top-down abductive learning procedures. Also, we do not have to compute eventually all answer-sets to find out if an atom is unknown, and we've introduced a method to choose one concept when the other is undefined in the well-founded model, which they cannot replicate. Last but not least, answer-sets semantics is not cumulative, which implies that you cannot assert what you learn without the risk of changing the semantics of what you learned.

Several other authors have also addressed the task of learning rules with exceptions [7, 6]. In these frameworks, nonmonotonicity and exceptions are dealt with by learning logic programs with negation. In [7] 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 [15] which can be easily mapped into negation as default.

It is worth mentioning that the treatment of exceptions by means of the addition of a non-abnormality literal (as we and [14] do) to each rule is similar to the approach for declarative debugging followed in [16]. In order to debug a logic program, in [16] the authors first transform it by adding a different default literal to each rule. These literals are then used as assumptions of the correctness of the rule, to be possibly revised in the face of a wrong solution. The debugging algorithm determines the assumptions that led to the wrong solution, thus identifying the incorrect rules.

Non-abnormality literals can also be viewed as new abducible predicates, as done for instance in [11, 10, 13]. In particular, in [11, 10] the authors have considered the integration and cooperation of induction and abduction in order to learn Abductive Logic Programs (ALP) from (possibly) incomplete

background knowledge expressed as ALP in its turn. In order to make a rule for a target predicate p consistent, the rule is specialized by adding a new abducible literal $not_abnorm_i(\vec{X})$. Exceptions are then ruled out by assuming $abnorm_i(\vec{t})$ for them. These assumptions are then used to learn a definition for $abnorm_i$ that describes the class of exceptions. In this way, they are able to learn hierarchies of exceptions.

5 Conclusions

We have shown that Extended Logic Programs are an appropriate representation formalism for learning in a three-valued setting. In this case, one has to learn a definition for both the target and the opposite concept, by considering positive and negative examples as instances of two different disjoint classes.

With ELP we are able to represent opposite concepts and to deal with their interaction. Inconsistencies are dealt with differently according to the type of atom causing them. Unseen atoms are classified as unknown through the adoption of non-deterministic rules, while exceptions are dealt with by non-abnormality defaults.

The level of generality of the learned definitions also impact on their interaction. In particular, we can learn the Least General Solution or the Most General Solution for both the positive and negative concept. The generality of the solutions should be chosen independently for the two definitions, thus leading to four epistemological cases. Future work will be devoted to identify the type of learning problems more appropriate for each case.

References

[1] J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*, volume 1111 of *LNAI*. SV, 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] J.J. Alferes, T.C. Przymusinski, and L.M. Pereira. "Classical" negation in non-monotonic reasoning and logic programming. *Journal of Automated Reasoning*, 1, 1998.

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

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

[6] 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.

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

[8] J. Dix. A classification-theory of semantics of normal logic programs: I. strong properties. *Fundamenta Informaticae*, XXII(3):227–255, 1995.

[9] J. Dix. A classification-theory of semantics of normal logic programs: Ii. weak properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.

[10] F. Esposito, S. Ferilli, E. Lamma, P. Mello, M. Milano, F. Riguzzi, and G. Semeraro. Cooperation of abduction and induction in logic programming. In Peter Flach and Antonis Kakas, editors, *Abductive and Inductive Reasoning*, Pure and Applied Logic. Kluwer, 1998. Submitted for publication.

[11] F. Esposito, E. Lamma, D. Malerba, P. Mello, M. Milano, F. Riguzzi, and G. Semeraro. Learning abductive logic programs. In M. Denecker, L. De Raedt, P. Flach, and A. Kakas, editors, *Proceedings of the ECAI96 Workshop on Abductive and Inductive Reasoning*. Catholic University of Leuven, 1996.

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

[13] Katsumi Inoue. Learning abductive and nonmonotonic logic programs. In Peter Flach and Antonis Kakas, editors, *Abductive and Inductive Reasoning*, Pure and Applied Logic. Kluwer, 1998. Submitted for publication.

[14] 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.

[15] A.C. Kakas, P. Mancarella, and P.M. Dung. The acceptability semantics for logic programs. In *Proceedings of the 11th International Conference on Logic Programming*, 1994.

[16] L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *Proceedings of the 2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 316–330. MIT Press, 1993.

[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] L.M. Pereira, Aparício, and J.J. Alferes. Nonmonotonic reasoning with well founded semantics. In *Proceedings of the International Conference on Logic Programming ICLP91*, pages 475–489. The MIT Press, 1991.

[19] L.M. Pereira, Aparício, and J.J. Alferes. Non-monotonic reasoning with logic programming. *Journal of Logic Programming*, 17:227–263, 1993.

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