Stochastic Propositionalization for efficient Multi-Relational Learning

N. Di Mauro, T.M.A. Basile, S. Ferilli, and F. Esposito

Department of Computer Science, University of Bari, ITALY
{ndm, basile, ferilli, esposito}@di.uniba.it

Abstract. The efficiency of multi-relational data mining algorithms, addressing the problem of learning First Order Logic (FOL) theories, strongly depends on the search method used for exploring the hypotheses space and on the coverage test assessing the validity of the learned theory against the training examples. A way of tackling the complexity of this kind of learning systems is to use a propositional method that reformulates a multi-relational learning problem into an attribute-value one. We propose a population based algorithm that using a stochastic propositional method efficiently learns complete FOL definitions.

1 Introduction

The efficacy of induction algorithms has been shown on a wide variety of benchmark domains. However, current machine learning techniques are inadequate for learning in more difficult real-world domains. The nature of the problem can be of different type as noise in the descriptions and lack of data, but also the representation language exploited to describe the examples of the target concept.

It is well known that the choice of the right representation for a learning problem has a significant impact on the performance of the learning systems. In traditional supervised learning problems, the learner is provided with training examples typically represented as vectors of attribute-value. In most complex real-world domains, however, it is necessary to adopt a more powerful representation language, such as a FOL language, a natural extension of the propositional representation, that is able to describe any kind of relation between objects. Inductive Logic Programming (ILP) systems are able to learn hypotheses expressed with this more powerful language. However, this representation language allows a potentially great number of mappings between descriptions (relational learning), differently from the feature vector representation in which only one mapping is possible (propositional learning). The obvious consequence of such a representation is that both the hypotheses space to search and the coverage test to assess the validity of the learned hypotheses against positive (all covered) and negative (all rejected) examples result more costly than in the case of propositional one.

A possible solution is a reformulation of the original multi-relational learning problem in a propositional one [4, 3, 2, 5]. During the reformulation, a fixed set of structural features is built from relational background knowledge and the structural properties of the individuals occurring in the examples. In such a process,

each feature is defined in terms of a corresponding program clause whose body is made up of a set of literals derived from the relational background knowledge. When the clause defining the feature is called for a particular individual (i.e., if its argument is bound to some example identifier) and this call succeeds at least once, the corresponding boolean feature is defined to be true.

Alternative approaches avoid the reformulation process and apply propositionalization directly on the original FOL context: the relational examples are *flattened* by substituting them with all (or a subset of) their matchings with a pattern. Following this idea, [7, 9, 1] proposed a multi-instance propositionalization in which each relational example is reformulated in its multiple matchings with a pattern. After that, to each initial observation correspond many feature vectors and the search for hypotheses may be recasted in this propositional representation as the search for rules that cover at least one instance per positive observation and no instance of negative observations.

This work proposes a propositionalization technique in which the transposition of the relational data is performed by an *online* flattening of the examples. The proposed method is a population based algorithm that stochastically propositionalizes the training examples in which the learning phase may be viewed as a bottom-up search in the hypotheses space. The method is based on a stochastic reformulation of examples that, differently from other proposed propositionalization techniques, does not use the classical subsumption relation used in ILP. For instance, in PROPAL [1], each example E, described in FOL, is reformulated into a set of matchings of a propositional pattern P with E by using the classical θ-subsumption procedure, being in this way still bond to FOL context. On the contrary, in our approach the reformulation is based on an rewriting of the training examples on a fixed set of domain constants.

2 The proposed technique

In this section the underlying idea of the propositionalization technique along with its implementation in the Sprol system will be given.

Let e an example of the training set, represented as a Datalog[1] ground clause, and let $consts(e)$ the set of the constants appearing in e. One can write a new example e' from e by changing one or more constants in e, i.e. by renaming.

In particular, e' may be obtained by applying an antisubstitution[2] and a substitution[3] under Object Identity to e, $e' = e\sigma^{-1}\theta_{OI}$. In the Object Identity framework, within a clause, terms that are denoted with different symbols must be distinct, i.e. they must represent different objects of the domain.

Definition 1 (Renaming of an example). *A ground renaming of an example E, is obtained by applying a substitution $\theta = \{V_1/t_1, V_2/t_2, \ldots V_n/t_n\}$ to $E\sigma^{-1}$,*

[1] Horn clause language without function symbols with arity greater than 0.

[2] An antisubstitution is a mapping from terms into variables.

[3] A substitution θ is a finite set of the form $\{V_1/t_1, V_2/t_2, \ldots V_n/t_n\}$ $(n \geq 0)$ where in each *binding* V_i/t_i the V_i is a variable $(V_i \neq V_j)$ and each t_i is a term. θ is a *ground substitution* when each t_i is a ground term.

such that $\{V_1, V_2, \ldots V_n\} \subseteq vars(E\sigma^{-1})$, $\{t_1, t_2, \ldots t_n\}$ *are distinct constants of* $consts(E)$, *and* σ^{-1} *is an antisubstitution.*

In this way, we do not need to use the θ-subsumption test to compute the renamings of an example E, we just have to rewrite it considering the permutations of the constants in $consts(E)$.

2.1 Generalizing examples

In the general framework of ILP the generalization of clauses is based on the concept of *least general generalization* originally introduced by Plotkin. Given two clauses C_1 and C_2, C_1 generalizes C_2 (denoted by $C_1 \leq C_2$) if C_1 subsumes C_2, i.e. there exists a substitution θ such that $C_1\theta \subseteq C_2$.

In our propositionalization framework, a generalization C (a non-ground clause) of two positive examples E_1 and E_2 may be calculated by turning constants into variables in the intersection between a renaming of E_1 and a renaming of E_2. In order to obtain consistent intersections, it is important to note that all the renamings, for both E_1 and E_2, must be calculated on the same fixed set of constants. Hence, given $E_1, E_2, \ldots, E_n$ examples, the set C of the constants useful to build the renamings may be chosen equal to $C = \arg\max_{E_i}(|consts(E_i)|)$.

Furthermore, to avoid empty generalizations, the constants appearing in the head literal of the renamings must be taken fixed. More formally, let $ren(E, C)$ a generic renaming of an example E onto the set of constants C, a generalization G such that subsumes both E_1 and E_2 is $(ren(E_1, C) \cap ren(E_2, C))\sigma^{-1}$.

Example 1. Given two positive examples $E_1 : h(a) \leftarrow q(a,b), c(b), t(b,c), p(c,d)$ and $E_2 : h(d) \leftarrow q(d,e), c(d), t(e,f)$, let $C = \arg\max_{E_i}(|consts(E_i)|) =$
$= consts(E_1) = \{a, b, c, d\}$. A generalization G of E_1 and E_2 is
$G = (ren(E_1, C) \cap ren(E_2, C))\sigma^{-1} =$
$= (\{h(a), \neg q(a,b), \neg c(b), \neg t(b,c), \neg p(c,d)\} \cap \{h(a), \neg q(a,b), \neg c(a), \neg t(b,c)\})\sigma^{-1} =$
$= \{h(a), \neg q(a,b), \neg t(b,c)\}\sigma^{-1} = (h(a) \leftarrow q(a,b), t(b,c))\sigma^{-1} =$
$= h(X) \leftarrow q(X,Y), t(Y,Z)$ with $\sigma^{-1} = \{a/X, b/Y, c/Z\}$.

2.2 Covering examples

In the classical ILP setting, generalizations are evaluated on the training examples using the θ-subsumption as a covering procedure. Here, the covering test is based on a syntactic lazy matching more efficient than the θ-subsumption.

Given a generalization G and an example E, it is possible to prove that G subsumes E under OI iff exists a permutation $P(n, r) = (c_1, c_2, \ldots, c_r)$ of size r of the constants $consts(E)$, with $r = |vars(G)|$, $n = |consts(E)|$ and $r \leq n$, such that $G\theta \cap E = G$ with $\theta = \{V_1/c_1, V_2/c_2, \ldots, V_r/c_r\}$, $V_i \in vars(G), V_i \neq V_j$. In order to be complete, the procedure must prove the test $G\theta \cap E = G$ for all the permutations $P(n, r)$. However, we can make the test stochastic by randomly choosing a number α of all the possible permutations.

To reduce the set of possible permutations we can fix the associations for the variables in the head of the generalization G. In particular if

Algorithm 1 Sprol

Input: $\underline{E^+}$: pos exs; $\underline{E^-}$: neg exs; $\underline{\alpha}$: the parameter for neg coverage; $\underline{\beta}$: the parameter for pos coverage; $\underline{k}$: the dimension of the population; $\underline{r}$: number of restarts;
Output: the hypotheses h

$C = \arg\max_{E_i \in E^+}(|consts(E_i)|)$;
while $E^+ \neq \emptyset$ **do**
  select a seed e from E^+
  Population $\leftarrow ren(k, e, C)$; */* select k renamings of e */*
  PopPrec $\leftarrow$ Population; $i \leftarrow 0$;
  while $i < r$ **do**
    P $\leftarrow \emptyset$;
    for each element $v \in$ Population **do**
      for each positive example $e^+ \in E^+$ **do**
        $V_{e^+} \leftarrow ren(t, e^+, C)$; */* select t renamings of e^+ */*
        P $\leftarrow$ P $\cup \{u | u = v \cap w_i, w_i \in V_{e^+}\}$; */* generalization */*
    Population $\leftarrow$ P;
    / Consistency check */*
    for each negative example $e^- \in E^-$ **do**
      $V_{e^-} \leftarrow ren(\alpha, e^-, C)$; */* select α renamings of e^- */*
      for each element $v \in$ Population **do**
        if v covers an element of V_{e^-} **then** remove v from Population
    / Completeness check */*
    for each element $v \in$ Population **do** completeness$_v \leftarrow 0$;
    for each positive example $e^+ \in E^+$ **do**
      $V_{e^+} \leftarrow ren(\beta, e^+, C)$; */* select β renamings of e^+ */*
      for each element $v \in$ Population **do**
        if $\exists u \in V_{e^+}$ s.t. $u \cap v = v$ **then** completeness$_v \leftarrow$ completeness$_v + 1$;
    $i \leftarrow i + 1$;
    if |Population| $= 0$ **then**
      Population $\leftarrow$ PopPrec; */* restart with the previous population */*
    else
      leave in Population the best k generalizations only; PopPrec $\leftarrow$ Population;
  add the best element $b \in$ Population to h;
  remove from E^+ the positive examples covered by b

$G : h(V_1, V_2, \ldots, V_d) \leftarrow \ldots$ and $E : h(c_1, c_2, \ldots, c_d) \leftarrow \ldots$
then we can fix the associations $\{V_1/c_1, V_2/c_2, \ldots, V_d/c_d\}$, $d \leq r, n$ in all the generated permutations. Furthermore, we can ulteriorly reduce the set of permutations by taking into account the positions of the costants in the literals. Supposing $p(V_1, V_2, \ldots, V_k)$ be a literal of the generalization G. Then, all the constants that may be associated to V_i, $1 \leq i \leq k$, are all those appearing in position i in the literals p/k of the example E.

2.3 The algorithm

Algorithm 1 reports the sketch of the Sprol system, implemented in Yap Prolog 5.1.1, that incorporates ideas of the propositional framework we proposed. Sprol

is a population based algorithm where several individual candidate solutions are simultaneously maintained using a constant size population. The population of candidate solutions provides a straightforward means for achieving search diversification and hence for increasing the exploration capabilities of the search process. In our case, the population is made up of candidate generalizations over the training positive examples. In many cases, local minima are quite common in search algorithms and the corresponding candidate solutions are typically not of sufficiently high quality. The strategy we used to escape from local minima is a *restart strategy* that simply reinitializes the search process whenever a local minimum is encountered.

Sprol takes as input the set of positive and negative examples of the training set and some user-defined parameters characterizing its stochastic behavior. In particular, α and β represent the number of renamings of a negative, respectively positive, example to use for the covering test; k is the size of the population; and, r is the number of restarts.

As reported in Algorithm 1, Sprol tries to find a set of clauses that cover all the positive examples and no negative one, by using an iterative population based covering mechanism. It sets the initial population made up of k randomly chosen renamings of a positive example (lines 3-4). Then, the elements of the population are iteratively generalized on the positive examples of the training set (lines 9-11). All the generalizations that cover at least one negative example are taken out (lines 14-17), and the quality of each generalization, based on the number of covered positive examples, is calculated (lines 18-24). Finally, best k generalizations are considered for the next iteration (line 28). In case of an empty population a restart is generated with the previous population (line 26).

Renamings of an example are generate randomly choosing a number of k renamings of the example E onto the set of its constants C.

It is important to note that our approach constructs hypotheses that are only approximately consistent. Indeed, in the consistency check it is possible that there exists a matching between an hypothesis and a negative example. The number α of allowed permutations is responsible of the induction cost as well as the consistency of the produced hypotheses. An obvious consequence is that the more permutations allowed, the more consistent the hypotheses found and, perhaps, the more learning time.

3 Discussion and Conclusion

In order to evaluate the system Sprol, we performed experiments on the classical mutagenesis dataset [8],consisting of structural descriptions of molecules to be classified into mutagenic and non-mutagenic ones, for which we considered the *atom bond* descriptions, the *indicator* variables, *logp* and *lumo*. The size k of the population has been set to 50, at the same way of α and β, and making 5 restarts.

As measures of performance, we use predictive accuracy and execution time. Results have been compared to that obtained by Progol [6]. The experiments

were performed exploiting a 10-fold cross-validation. The Sprol results, averaged over the 10-folds, show an improvement of the execution time with respect to Progol (56.35 sec. of Sprol vs 546.25 sec. of Prolog) obtaining a good predictive accuracy of the learned theory (80.4% of Sprol vs 79.81% of Prolog).

As a concluding remark, the proposed population based algorithm is able to efficiently solve multi-relational problems by using a stochastic propositional method. The result of an empirical evaluation on the mutagenesis dataset of the proposed technique is very promising and proves the validity of the method.

It is important to note that α and β parameters used in the algorithm for checking consistency and completeness lead to different behaviors of the induction process. In particular, in complex domains with a lot of failure derivations between hypotheses and negative examples, low values for α may lead to inconsistent hypotheses. On the other way, low values for β may produce a theory with many hypotheses. An extensive study of this problem is needed and it represents an important future work. Furthermore, we plan to automatically discover, in an online manner, the correct input parameters of Sprol for a given learning task.

Finally, more experiments, and comparisons with other classical ILP systems, are needed to better evaluate the methodology, especially using synthetic data well suited for a parameter setting study.

References

1. E. Alphonse and C. Rouveirol. Lazy propositionalization for relational learning. In W. Horn, editor, *Proceedings of ECAI'2000*, pages 256–260. IOS Press, 2000.
2. M.-A. Krogel, S. Rawles, F. Zelezny, P. Flach, N. Lavrac, and S. Wrobel. Comparative evaluation of approaches to propositionalization. In T. Horvath and A. Yamamoto, editors, *Proceedings of ILP'2003*, volume 2835 of *Lecture Notes in Computer Science*, pages 194–217. Springer-Verlag Heidelberg, 2003.
3. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, 1994.
4. N. Lavrac, S. Dzeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff, editor, *Machine Learning - EWSL*, volume 482 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 1991.
5. N. Lavrač, F. Železný, and P. A. Flach. RSD: Relational subgroup discovery through first-order feature construction. In S. Matwin and C. Sammut, editors, *Proceedings of ILP'2002*, volume 2583 of *LNAI*, pages 149–165. Springer Verlag, 2002.
6. S. Muggleton. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
7. M. Sebag and Rouveirol C. Tractable induction and classification in first order logic via stochastic matching. In *Proceedings of IJCAI'97*, pages 888–893, 1997.
8. A. Srinivasan, S. Muggleton, and R.D. King. Comparing the use of background knowledge by inductive logic programming systems. In L. De Raedt, editor, *Proceedings of ILP'95*, pages 199–230. Springer-Verlag Heidelberg, 1995.
9. J.-D. Zucker and J.-G. Ganascia. Representation changes for efficient learning in structural domains. In *Proceedings of ECML'96*, pages 543–551, 1996.