A Link-Based Method for Propositionalization

Quang-Thang DINH, Matthieu EXBRAYAT, Christel VRAIN

LIFO, Bat. 3IA, Université d'Orléans
Rue Léonard de Vinci, B.P. 6759, F-45067 ORLEANS Cedex 2, France
{Thang.Dinh,Matthieu.Exbrayat,Christel.Vrain}@univ-orleans.fr
http://www.univ-orleans.fr/lifo/

Abstract. Propositionalization, a popular technique in Inductive Logic Programming, aims at converting a relational problem into an attribute-value one. An important facet of propositionalization consists in building a set of relevant features. To this end we propose a new method, based on a synthetic representation of the database, modeling the links between connected ground atoms. Comparing it to two state-of-the-art logic-based propositionalization techniques on three benchmarks, we show that our method leads to good accuracy in supervised classification.

1 Introduction

Propositionalization is a popular technique in ILP, that aims at converting a relational problem into an attribute-value one [1–5]. Propositionalization usually decomposes into two main steps: generating a set of useful attributes (features) starting from relational representations and then building an attribute-value table, which can be mono-instance (a single tuple for each example) or multi-instance (several tuples for an example). Traditional attribute-value algorithms can then be applied to solve the problem. Approaches for constructing automatically the new set of attributes (features) can be divided into two trends [6, 7]: methods based on logic *or* inspired from databases.

The first trend follows the ILP tradition which is logic-based. This trend, as far as we know, includes the first representative LINUS system [8] and its descendants, the latest being RSD [9], HiFi [4] and RelF [5]. For these systems, examples are mostly represented as first-order Herbrand interpretations and features are conjunctions of first-order function-free atoms. The search for features is based on a *template* (a set of ground atoms of which all arguments fall in exactly one of two categories: "input" or "output") or mode declarations (defining the predicates and assigning a *type* and *mode* to each of their arguments).

The second trend is inspired from databases and appeared later beginning with systems like Polka [2], RELAGGS [10] and RollUp [7]. Those systems build attributes, which summarize information stored in non-target tables by applying usual database aggregate functions such as count, min, max, etc. The method in RELAGGS is very similar to the one in Polka developed independently by a different research group. A difference between them concerns efficiency of the implementation. Besides the focus on aggregation functions, RELAGGS concentrates on the exploitation of relational database schema information, especially foreign key relationships as well as the use of optimization techniques such as indices, which are frequently used in relational databases.

In this paper, we propose a new method, called *Link-Based Propositionalization* or LBP, to build features for propositionalization from a set of ground atoms, without information on templates or mode declarations. The method was initially designed to learn the structure of Markov logic networks [11], where it was used as a strategy to build a boolean table and to find dependent literals. The originality of the method is to build an abstract representation of sets of connected ground atoms, allowing thus to represent properties between objects.

LBP differs from the classical logic-based approaches both in the *semantic of the boolean table* and in the *search for features.* For example, the RelF system uses a block-wise technique to construct a set of tree-like conjunctive relational features while the others, like HiFi or RSD use the traditional level-wise approaches. The search in LBP does not rely on template or mode declarations, but on a synthetic representation of the dataset, namely the links of the chains, which allows to build features as well as to construct the boolean table based on the regularities of these chains. The notion of chain is related to relational path-finding [12] and relational cliché [13].

Our propositional method is presented in Section 2. Section 3 is devoted to experiments and finally, Section 4 concludes this paper.

2 Link-Based Propositionalization

We suppose the reader familiar with the notions of atoms, literals, ground atoms and clauses. Here a *variable literal* denotes a literal that contains only variables. Two ground atoms are *connected* if they share at least one constant.

The method that we propose is based on an abstract representation of sets of connected atoms, either ground atoms or variable atoms. This abstract representation is learned from sets of connected ground atoms and it is used to build sets of connected variable literals. Let us first introduce this representation.

2.1 An abstract representation

The idea underlying this method is to detect regularities in ground atoms: we expect that many chains of connected atoms are similar, and could thus be variabilized by a single chain and that only the set of variable literals appearing in this chain has to be stored. The representation that we propose is based on the notion of links, which models the relations between connected atoms.

Definition 1. *Let* g *and* s *be two ground literals (resp. two variable literals). A* link *between* g *and* s *is a list composed of the name of the predicates of* g *and* s *followed by the positions of the shared constants (resp. variables). It is written* link(g, s) = {G S g_0 s_0 / g_1 s_1 / ... } *where* G *and* S *are the predicate symbols of* g *and* s, $g_i \in [1, arity(g)]$, $s_i \in [1, arity(s)]$ *and the combinations* / g_i s_i / *mean that the constants respectively at position* g_i *in* g *and* s_i *in* s *are the same. If* g *and* s *do not share any constant then* link(g,s) *is empty.*

We are interested in representing the properties of sets of connected literals. In order to have a sequential representation of these properties, we consider only chains of literals defined as follows:

Definition 2. *A* chain *of ground literals (resp. variable literals) starting from a ground (resp. variable) literal* g_1 *is a list of ground (resp. variable) literals* $\langle g_1, ..., g_k, ...\rangle$ *such that* $\forall i > 1$, $link(g_{i-1}, g_i)$ *is not empty and every constant (resp. variable) shared by* g_{i-1} *and* g_i *is not shared by* g_{j-1} *and* $g_j, 1 < j < i$. *It is denoted by* $chain(g_1) = \langle g_1, ..., g_k, ...\rangle$. *The length of the chain is the number of atoms in it.*
The link of the chain $\langle g_1, ..., g_k, ...\rangle$ *is the ordered list of links* $link(g_i, g_{i+1})$, $i \geq 1$, *denoted by* $link(gc) = \langle link(g_1, g_2)/.../link(g_i, g_{i+1})/...\rangle$.

Definition 3. *A link* $\langle g_1, ..., g_k\rangle$ *is said to be a prefix of another link* $\langle s_1, ..., s_n\rangle$, *if there exists* $k \leq n$ *such that* $link(g_i, g_{i+1}) = link(s_i, s_{i+1})$, $\forall i, 1 \leq i < k$.

Example 1. Let $G = \{$*P(a, b)*, *Q(b, a)*, *R(b, c)*, *S(b)*, *S(c)*$\}$ be a set of ground atoms. We have: *link(P(a, b), Q(b, a))* = $\{$*P Q 1 2 / 2 1*$\}$. A possible *chain* starting from the ground atom *P(a, b)* is $\langle$*P(a, b)*, *R(b, c)*, *S(c)*$\rangle$.

The link of chain $\langle$*P(a, b)*, *R(b, c)*, *S(c)*$\rangle$ is $\langle\{$*P R 2 1* $\}$ / $\{$*R S 1 1*$\}\rangle$. The link of chain $\langle$*P(a, b)*, *R(b, c)*$\rangle$ is $\langle\{$*P R 2 1* $\}\rangle$, which is a prefix of the previous link. On the other hand, $\langle$*P(a, b)*, *R(b, c)*, *S(b)*$\rangle$ is not a chain as the constant *b* share by *R(b, c)* and *S(b)* is already used to link *P(a, b)* and *R(b, c)*.

2.2 Creation of a set of features

Let us consider a target predicate *P* and a training dataset *DB*. We aim at building a set *SL* of variable literals linked to *P* given *DB* such that for each true ground atom *e* built with predicate *P* in *DB*, and for each chain *chain(e)* starting from *e*, there exists a variabilization such that $var(chain(e)) \subseteq SL$.

The algorithm can be sketched as follows (in practice the length of the chains is limited by an integer *k*)
• for each true ground atom *A* of the target predicate *P*,
  • find every chain starting from *A*
  • build the corresponding link and check whether it is a prefix of a link already built
  • if not, variabilize it: a *simple variabilization strategy* is used ensuring that different constants in this *chain* are replaced by different variables.

Example 2. Let *DB* be a database composed of 14 ground atoms as follows: *advisedBy(Bart, Ada), student(Bart), professor(Ada), publication(T1,Bart), publication(T2, Bart), publication(T1, Ada), publication(T2, Ada), advisedBy(Betty, Alan), student(Betty), professor(Alan), publication(T3, Betty), publication(T3,Alan), publication(T3, Andrew), professor(Andrew).*

Figure 1 illustrates the production of chains of ground literals with the corresponding links and the resulting variable literals, using *advisedBy* as the target predicate, and bounding the length of chains to 4.

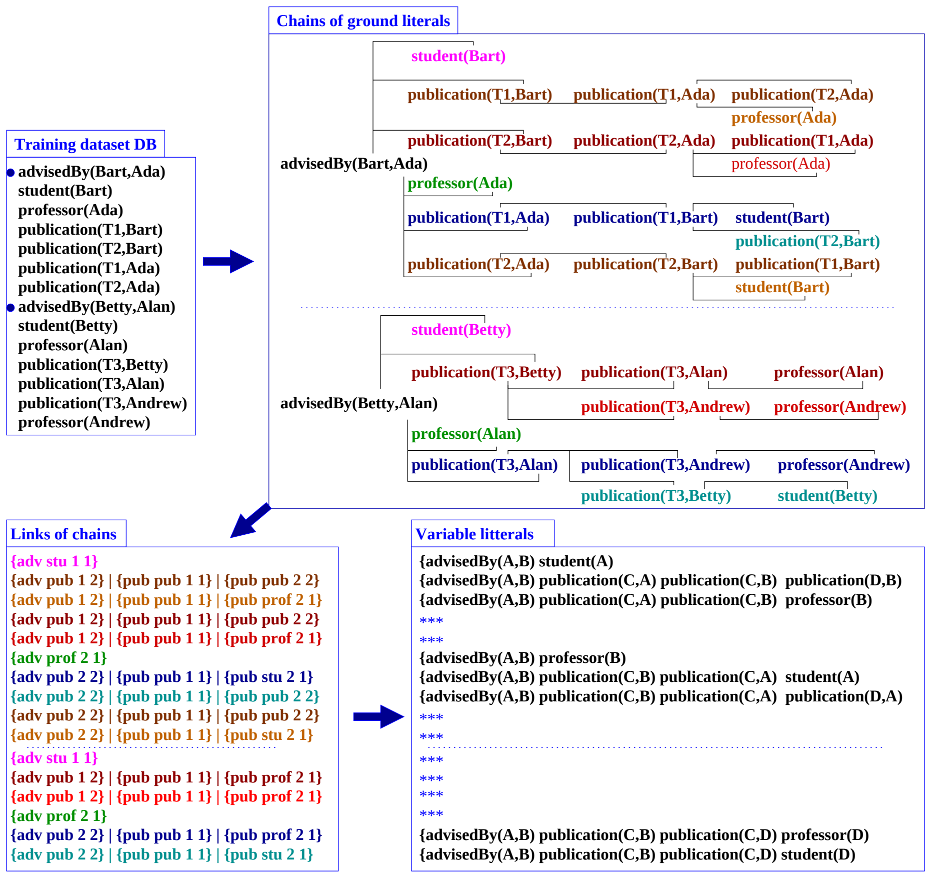
Training dataset DB
● advisedBy(Bart,Ada)
student(Bart)
professor(Ada)
publication(T1,Bart)
publication(T2,Bart)
publication(T1,Ada)
publication(T2,Ada)
● advisedBy(Betty,Alan)
student(Betty)
professor(Alan)
publication(T3,Betty)
publication(T3,Alan)
publication(T3,Andrew)
professor(Andrew)
Chains of ground literals
advisedBy(Bart,Ada)
student(Bart)
publication(T1,Bart) publication(T1,Ada) publication(T2,Ada) professor(Ada)
publication(T2,Bart) publication(T2,Ada) publication(T1,Ada) professor(Ada)
professor(Ada)
publication(T1,Ada) publication(T1,Bart) student(Bart) publication(T2,Bart)
publication(T2,Ada) publication(T2,Bart) publication(T1,Bart) student(Bart)
advisedBy(Betty,Alan)
student(Betty)
publication(T3,Betty) publication(T3,Alan) professor(Alan)
publication(T3,Andrew) professor(Andrew)
professor(Alan)
publication(T3,Alan) publication(T3,Andrew) professor(Andrew)
publication(T3,Betty) student(Betty)
Links of chains
{adv stu 1 1}
{adv pub 1 2} | {pub pub 1 1} | {pub pub 2 2}
{adv pub 1 2} | {pub pub 1 1} | {pub prof 2 1}
{adv pub 1 2} | {pub pub 1 1} | {pub pub 2 2}
{adv pub 1 2} | {pub pub 1 1} | {pub prof 2 1}
{adv prof 2 1}
{adv pub 2 2} | {pub pub 1 1} | {pub stu 2 1}
{adv pub 2 2} | {pub pub 1 1} | {pub pub 2 2}
{adv pub 2 2} | {pub pub 1 1} | {pub pub 2 2}
{adv pub 2 2} | {pub pub 1 1} | {pub stu 2 1}
{adv stu 1 1}
{adv pub 1 2} | {pub pub 1 1} | {pub prof 2 1}
{adv pub 1 2} | {pub pub 1 1} | {pub prof 2 1}
{adv prof 2 1}
{adv pub 2 2} | {pub pub 1 1} | {pub prof 2 1}
{adv pub 2 2} | {pub pub 1 1} | {pub stu 2 1}
Variable litterals
{advisedBy(A,B) student(A)
{advisedBy(A,B) publication(C,A) publication(C,B) publication(D,B)
{advisedBy(A,B) publication(C,A) publication(C,B) professor(B)
***
***
{advisedBy(A,B) professor(B)
{advisedBy(A,B) publication(C,B) publication(C,A) student(A)
{advisedBy(A,B) publication(C,B) publication(C,A) publication(D,A)
***
***
***
***
***
***
{advisedBy(A,B) publication(C,B) publication(C,D) professor(D)
{advisedBy(A,B) publication(C,B) publication(C,D) student(D)

Fig. 1. The variabilization process using *chains* and *links* (length ≤ 4)

2.3 Construction of a boolean table

The next step in our approach transforms information in the database into a boolean table BT, where each column corresponds to a variable literal and each row corresponds to a true/false ground atom of the target predicate. Let us assume that data concerning a given ground atom q_r is stored in row r. Let us also assume that column c corresponds to a given variable literal vl_c. $BT[r][c]$ = true means that starting from q_r we can reach a literal that is variabilized as vl_c. In other words, there exists at least a variabilized chain $vc \in SL$ containing the variable literal vl_c, a ground chain gc_r starting from the ground atom q_r, and a variabilization of gc_r such that $vc \subseteq var(gc_r)$. Let us notice that to find information related to a variable literal we only need to consider the subset of variable literals appearing in the set of chains starting from this literal, which is much smaller than the complete set SL, especially when the database is large.

3 Experiments

Systems, Databases and Methodology
We evaluated LBP according to classification accuracy, which has been used in most of previous research as in [4, 5, 7, 9]. We compared LBP to two state-of-the-art logic-based systems: RelF [5] and RSD [9]. For the comparison of logic-based and database-inspired methods, we refer to [6, 7, 14] for further reading.
We performed experiments on three domain-popular datasets:
• *Mutagenesis* concerns drugs : their chemical properties, their atoms and bonds. It consists of 188 organic molecules marked according to their mutagenicity. The learning task is to predict whether a drug is mutagenic or not.
• *UW-CSE* describes an academic department (15 predicates; 1323 constants; 2673 ground atoms). We have chosen to predict who is the advisor of who.
• *CORA* consists of citations of computer science papers (10 predicates; 3079 constants; 70367 true/false ground atoms). We learned the predicate *sameBib*.
We conducted a 10-fold cross-validation. Accuracy was computed using two decision tree classifiers of WEKA [15]: J48 and REPTree, that received as input the features generated by each system. We used the same feature declaration bias for RSD and RelF and allowed cyclic features in RSD [5]. For LBP, we set the maximal length of considered g-chains (v-chains) to $k = 4$.

Results
Table 1 indicates, for each dataset and propositionalization method, the number of features generated (NoF) and the running time (T) of the method, together with the resulting predictive accuracy with J48 and REPTree (REP).

LBP achieved the best results on Mutagenesis and UW-CSE, with both classifiers, and performed a little worse than RelF on CORA. RSD did not finish on the largest dataset CORA. We can notice that J48 did globally perform a little worse than REPTree on all of our experiments.

	LBP				RelF				RSD			
	NoF	T	J48	REP	NoF	T	J48	REP	NoF	T	J48	REP
Mutagenesis	43	**1.2**	75.6	**79.7**	455	**1.2**	73.1	76.2	321	1.3	71.9	72.1
UW-CSE	102	10.5	80.2	**83.9**	567	**10.3**	80.2	81.3	491	11.2	70.7	75.7
CORA	41	**31.7**	83.3	85.0	347	32.5	83.4	**85.7**	DNF	DNF	DNF	DNF

Table 1. Accuracy results

For all datasets, LBP did generate much less features than RelF and RSD. Beside the impact on the classifier input, this number of features does also influence the size of the boolean table. However, considering CORA, despite the smaller number of learned features, the size of the boolean table learned by LBP (6.3MB) is much larger than the one learned by RelF (4.6MB). This is because the number of rows in LBP depends on the number of ground atoms of the query predicate (which are numerous in CORA) whereas it depends on the number of true combinations of variables in RelF. Yet, running-times remain similar.

Based on these results, we can conclude that our system is competitive to the state-of-the-art propositional systems on these three benchmark datasets.

4 Conclusion and Future Work

We introduced LBP, a link-based method of propositionalization to transform relational data into a boolean table. Experiments show that LBP can lead to a better accuracy of classification while generating a smaller number of features than state-of-the-art methods. LBP does currently only support binary classification, as its boolean tables are based on the true/false ground atoms (of the query predicate). Handling multi-class problems is thus a near future goal.

References

1. Alphonse, É., Rouveirol, C.: Selective propositionalization for relational learning. In: PKDD'99. Volume 1704 of LNCS, Springer (1999) 271–276
2. Knobbe, A.J., de Haas, M., Siebes, A.: Propositionalisation and aggregates. In: PKDD'01. Volume 2168 of LNCS, Springer (2001) 277–288
3. De Raedt, L.: Logical and Relational Learning. Springer (2008)
4. Kuželka, O., Železný, F.: Hifi: Tractable propositionalization through hierarchical feature construction. In: Late Breaking Papers, ILP'08. (2008)
5. Kuželka, O., Železný, F.: Block-wise construction of tree-like relational features with monotone reducibility and redundancy. Mach. Learn. **83**(2) (2011) 163–192
6. Krogel, M.A., Rawles, S., Železný, F., Flach, P.A., Lavrac, N., Wrobel, S.: Comparative evaluation of approaches to propositionalization. In: ILP'03. Volume 2835 of LNCS, Springer (2003) 197–214
7. Lesbegueries, J., Lachiche, N., Braud, A.: A propositionalisation that preserves more continuous attribute domains. In: ILP'09. (2009)
8. Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood (1994)
9. Lavrac, N., Zelezný, F., Flach, P.A.: Rsd: Relational subgroup discovery through first-order feature construction. In: ILP'02. Volume 2583 of LNCS, Springer (2002) 149–165
10. Krogel, M.A., Wrobel, S.: Transformation-based learning using multirelational aggregation. In: ILP'01. Volume 2157 of ILP, Springer (2001) 142–155
11. Dinh, Q.T., Exbrayat, M., Vrain, C.: Discriminative markov logic network structure learning based on propositionalization and chi^2-test. In: ADMA'10. Volume 6440 of LNCS, Springer (2010) 24–35
12. Richards, B.L., Mooney, R.J.: Learning relations by pathfinding. In: AAAI'92, AAAI Press / The MIT Press (1992) 50–55
13. Silverstein, G., Pazzani, M.J.: Relational clichés: Constraining induction during relational learning. In: ML'91, Morgan Kaufmann (1991) 203–207
14. Kuzelka, O., Zelezný, F.: Block-wise construction of acyclic relational features with monotone irreducibility and relevancy properties. In: ICML'09, ACM (2009) 72
15. Machine Learning Group at University of Waikato: Data mining software in java. `http://www.cs.waikato.ac.nz/ml/weka/`