Stochastic Propositionalization of Non-Determinate Background Knowledge

Stefan Kramer
Austrian Research Institute for Artificial Intelligence
Schottengasse 3
A-1010 Vienna, Austria
stefan@ai.univie.ac.at

April 10, 1997

Abstract

It is a well-known fact that propositional learning algorithms require "good" features to perform well in practice. So a major step in data engineering for inductive learning is the construction of good features by domain experts. These features often represent properties of structured objects, where a property typically is the occurrence of a certain substructure having certain properties. To partly automate the process of "feature engineering", we devised an algorithm that searches for features which are defined by such substructures. The algorithm stochastically conducts a top-down search for first-order clauses, where each clause represents a binary feature. It differs from existing algorithms in that its search is not class-blind, and that it is capable of considering clauses ("context") of almost arbitrary length (size). Preliminary experiments are favorable, and support the view that this approach is promising.

1 Introduction

A very large number of algorithms require a propositional representation, whereas many real-world learning problems are essentially relational. To bridge this gap, various researchers (e.g., [10]) have proposed a transformation approach. This type of transformation is called propositionalization.

In general, full equivalence between the original and the transformed problem can only be achieved for certain subsets of first-order logic and certain types of background knowledge ([10], [2], [3]). But even if there could be equivalent transformations theoretically, most interesting cases would still require feature subset selection. So for pragmatic reasons we should not expect the transformed problem to be equivalent to the original problem. Note that such a transformation may also be viewed as *constructive induction*, i.e., as a representation change for learning.

It should be clear that the *problems* of propositionalization and of concept learning are not the same, even if there might be some similarities in the *solutions* to these problems.

First of all, constructing new features should best be viewed as a preparatory step prior to learning. Propositional learning algorithms heavily rely on "good" features. Secondly, most learning tasks involve solving the set-cover problem. During feature construction we do not have to bother about coverage. Rather, we have to strive for features that can be used to create good *partitionings* of the instance space. Thirdly, concept learning is supervised, while feature construction can be supervised or unsupervised (concept formation). Fourthly, in practice we often know some expert-provided features (such as the so-called "Hansch attributes" or "structural alerts" known from chemical domains), and we are interested in finding features that complement them. This is also specific to propositionalization.

Since we have to select from a large number of features, we have to specify preferences. We determined the following requirements that should be fulfilled by features constructed from background knowledge. The features should be

- **(R1)** not too specific and not too general.
- **(R2)** not too complex. (So they have at least a potential for being comprehensible.)
- **(R3)** different from one another.
- **(R4)** different from existing, expert-provided features.

In the next section we describe a new algorithm for propositionalization, and relatively straight-forward solutions to the above requirements. In section 3, we will present a few results from preliminary experiments. In section 4, we will review related work.

2 Description of the Method

In this section we describe a new algorithm for propositionalization. The algorithm stochastically conducts a top-down search for first-order clauses, where each clause represents a binary feature. It differs from existing algorithms in that its search is not class-blind, and that it is capable of considering clauses ("context") of almost arbitrary length (size).

First we give a brief overview of the algorithm as it is described in figure 1. In the following, the algorithm will be referred to as SP.

The algorithm maintains three sets of clauses. One set (*CurrentClauses*) contains the clauses of the current generation. This set (of fixed size *NrClauses*) is developed over a predefined number of generations (*NrGenerations*). Another set of what we will call "parent clauses" (*ParentClauses*) contains the clauses which are the basis for specializations. The third set (*BestClauses*) is simply the set of the best clauses found so far.

The algorithm proceeds in the following way: in each step, the *NrClausesToReplace* "stochastically" worst clauses are removed from the current clauses, and the same number of new clauses is generated stochastically. The probability of removing a clause is proportional to the inverse of its evaluation.

The algorithm that generates new clauses can be found in figure 2: it randomly selects a parent from the set of potential parents with a probability proportional to the *expected evaluation of all possible refinements*. Subsequently, a literal is selected for the specialization of the chosen parent clause with a probability proportional to the evaluation of the resulting clause. The refinement operator is a kind of specialization using schemata [15]. Note that the only operator used in the algorithm is a refinement operator. Currently, the evaluation function employed is the inverse of the chi-square statistic. Due to lack of space, we cannot describe the expected frequencies here in detail. However, the basic approach is independent of the used evaluation function.

Every generated clauses is a candidate for being a parent clause, but only a subset of parent clauses is actually kept, since the chance of selecting a good parent clause decreases with an increase in the number of parent clauses. Only those candidate parent clauses with the best expected evaluations of refinements are kept. In contrast to all other parent clauses, the most general clause can never be discarded. This enables recovery from too many overly specific clauses in the population. (Remember that the only operator used is a refinement operator.)

One key feature of the algorithm is the distinction between "parent clauses" and "current clauses". The reason for this is that "being a good child is *not* being a good parent." More technically, a clause with a good evaluation *not necessarily* has good refinements, and vice versa, a clause producing good refinements *not necessarily* has to have a good evaluation.

After each step, the current set of clauses is compared with the best set of clauses so far. If the current set is better than the best set so far, it is remembered as the new best set so far. After *NrGenerations* generations of clauses, the best set of clauses up to then is returned.

Sets of clauses are evaluated in the following way: the features corresponding to the given clauses are used to create the finest possible partitioning of the training instances that can be obtained by these features. For each partition, we predict the majority class of the instances contained. Comparing the predicted values with the observed values, we calculate the inverse of the chi-square statistic as for single clauses.

The type of search presented here is not a genetic algorithm, since cross-over

```
procedure stochastic_propositionalization (Target, NrGenerations, NrClauses,
   NrClausesToReplace, NrParentClauses)

Input:
Target:               literal with target predicate
NrGenerations:        number of steps to be performed
NrClauses:            number of clauses in the set of current clauses
                      (the "population")
NrClausesToReplace:   number of clauses to be replaced from the set of
                      current clauses in each step
NrParentClauses:      number of parent clauses to keep

Output:
BestClauses:          a set of clauses corresponding to binary features
                      constructed from non-determinate background
                      knowledge

CurrentClauses ← randomly_refine_parent_clauses(NrClauses,
      {Target :- true})
ParentClauses ← select_best_parent_clauses(NrParentClauses − 1,
      CurrentClauses)
    ∪{Target :- true}
BestClauses ← CurrentClauses

for i ← 2 to NrGenerations do
  CurrentClauses ← randomly_remove_clauses(NrClausesToReplace,
        CurrentClauses)
  /*
    Randomly remove clauses from the set of current clauses, with a
    probability proportional to the inverse of the evaluation of the
    respective clause.
  */

  NewClauses ← randomly_refine_parent_clauses(NrClausesToReplace,
        ParentClauses)
  CurrentClauses ← CurrentClauses ∪ NewClauses
  ParentClauses ← select_best_parent_clauses(NrParentClauses − 1,
       ParentClauses \ {Target :- true} ∪ NewClauses)
    ∪{Target :- true}

  if evaluation(CurrentClauses) > evaluation(BestClauses) then
       BestClauses ← CurrentClauses

return BestClauses
end procedure
```

Figure 1: Pseudocode of the algorithm for stochastic propositionalization.

```
procedure randomly_refine_parent_clauses(NrNewClauses, ParentClauses)

Input:
NrNewClauses:     number of new clauses to be generated as refinements
                  of some of the given parent clauses
ParentClauses:    clauses used as basis for refinements

Output:
NewClauses:       the newly generated clauses

NewClauses ← {}

for i ← 1 to NrNewClauses do
  ParentClause ← randomly_select_parent_clause(ParentClauses)
  /*
    Selects a parent clause with a probability proportional to the
    expected evaluation of all possible children.
  */

  NewClause ← randomly_refine_clause(ParentClause)
  /*
    Selects a refinement of the chosen parent clause with a probability
    proportional to the evaluation of the resulting clause.
  */

  NewClauses ← NewClauses ∪ {NewClause}
return NewClauses
end procedure
```

Figure 2: Pseudocode of the procedure for random refinements of given parent clauses.

and mutation operators are missing. A full-fledged genetic algorithm would also need a generalization operator in addition to the specialization operator.

The overall approach would not work, if the clauses in the population were the same extensionally. In other words, there would be no "division of labour" among the clauses (**R3**). (This is also the motivation for the universal suffrage selection algorithm presented in [6].) We took a simple *extension-driven approach* to solve this problem: the algorithm only considers those refinements that yield clauses with an extension different from the extensions of clauses in the current population. This (extensional) restriction can also be used to enforce the construction of features that are different from expert-provided features (**R4**). As expert-provided features are a form of established knowledge, the extension-driven approach enables discoveries: the new features have to complement the existing ones.

Finally, the rest of the above requirements are addressed: **R1** is realized by parameters for required minimum and maximum coverage of clauses. **R2** requires restrictions concerning the complexity of clauses. First, we restrict the maximum number of variables of certain user-defined types, as the number of variables is crucial for the comprehensibility and for the feasibility of search in general. Second, the number of literals can be restricted.[1] Third, no negation is used in the clauses, since it often is detrimental to comprehensibility.

3 Experimental Results

3.1 Family Domain

We conducted experiments in the family domain [12] with the relations `son(A,B)` and `niece(A,B)`. Although the family domain is determinate, it helped us see which things work and which do not.

SP consistently found correct solutions for both relations. (However, due to the probabilistic nature of the algorithm, there is no guarantee for that.) Here are a few features found for the relation `son(A,B)`:

```
new_f1(A, B) :- father(B, A),
                brother(A, _).

new_f2(A, B) :- mother(B, A),
                husband(A, _).

new_f3(A, B) :- nephew(A, C),
                brother(B, C).
```

Note that the predicate `male` is not available, so that the algorithm tries to approximate it by `brother(A,_)`, `husband(A, _)` and `nephew(A, C)`. Also note that `new_f3` is not the easiest way to express this relation!

Since the family domain is very simple, all the work is already done in the propositionalization step. In most real-world domains, however, the chance of finding the correct concept during propositionalization is very small. So usually the work is divided by the propositionalization algorithm and by the subsequently applied learning algorithm.

3.2 Carcinogenicity Domain

Next, we performed experiments in the carcinogenicity domain [7]. The database contains information about the carcinogenicity of 330 compounds, as classi-

[1] So in fact we do not allow for clauses of *arbitrary* length, but still the bounds used are far too large for considering all clauses up to this length.

fied by the US National Institute of Environmental Health Sciences (NIEHS). Chemicals are classified as carcinogenic or not (compounds classified as equivocal are considered non-carcinogenic). The chemical structures are described at the atom and bond level, and in terms of various relevant structural properties: functional groups (such as benzene rings and methyl groups) and "structural alerts". Structural alerts are structural properties which have been identified by domain experts as indicators of carcinogenicity. Structural alerts mostly consist of functional groups.

Our goal was to find structural alerts other than those already known to domain experts. So in this application SP searched the space of combinations of functional groups with the restriction that the corresponding features should be (extensionally) different from the known structural alerts. (The parameters used were $NrGenerations = 50$, $NrClauses = 5$, $NrClausesToReplace = 1$, $NrParentClauses = 10$.) SP found several interesting combinations of functional groups. This is an example of a combination that was found to be deactivating:

```
new_f4(A) :- functional_group(A, B, methyl),
             connected(B, C),
             functional_group(A, C, ring_size_5).
```

After propositionalization, we applied C4.5 [14].

In table 1, we summarize the results for various methods in this domain. T2 [1] induces 2-level decision trees. FOIL [12] and Progol[11][2] are state-of-the-art ILP algorithms. M5 [13] learns regression trees with linear regression models in the leaves. SRT [9] learns relational regression trees. The propositional learning algorithms listed here utilize global features available in addition to the non-determinate background knowledge. Quantitatively, SP/C4.5 performs quite well, and the improvement over other propositional algorithms is due to the newly constructed features. The only algorithms that significantly perform better (M5, SRT) are those which utilize the additional information given in the formulation as a regression problem.

4 Related Work

In this section we briefly review related work on propositionalization and stochastic search in machine learning and Inductive Logic Programming.

LINUS [10] was the first system to transform a relational representation into a propositional representation. The hypothesis language of LINUS is restricted to function-free constrained DHDB (deductive hierarchical database) clauses. This implies that no recursion is allowed, and that no new variables may be introduced.

[2] The experiment with Progol has been described in [7].

Method	Accuracy
Default	55.00%
Ames Test	63.00%
C4.5 prune	58.79%
C4.5 rules	60.76%
T2	65.00%
M5	69.93%
FOIL	25.15%
Progol	63.00%
SRT	72.46%
SP/C4.5 prune	66.78%

Table 1: Quantitative results for the carcinogenicity domain obtained by 5-fold cross-validation.

DINUS [10] weakens the language bias of LINUS so that the system can learn clauses with a restricted form of new variables, namely determinate variables. This allows for the same transformation approach as the one taken in LINUS.

Cohen [2] introduces a new restriction on non-determinate free variables called "locality constraint". This can be thought of in terms of schemata or clichés [15] of literals. Newly introduced non-determinate variables may only be reused in literals within the same instantiated schema or cliché.

Turney [17] described a *special purpose* program for translating the "trains" problem of the East-West challenge into a propositional representation. Zucker and Ganascia [18] proposed to decompose structured examples into several learning examples, which are descriptions of parts of what they call the "natural example". Cohen [3] introduced the notion of "set-valued features", which can be used to transform certain types of background knowledge.

Srinivasan and King [16] presented a method for feature construction based on hypotheses returned by Progol [11]. For each clause, each input-output connected subset of literals is used to define a feature. In contrast to all previously discussed methods, this method works for all types of background knowledge. However, it is not yet clear why particularly these features should be useful for transforming relational learning problems.

Geibel and Wysotzki [5] propose a method for feature construction in a graph-based representation. The features are obtained through fixed-length paths in the neighborhood of a node in the graph. The constructed features are either "context-dependent node attributes of depth n" or "context dependent edge attributes of depth n". This method also works in general (for graphs), but using fixed-length paths obviously becomes prohibitive for large n.

In contrast to SP, REGAL [6] is a *concept learning* algorithm. It is a full-fledged genetic algorithm. REGAL's universal suffrage selection algorithm is the first extension-driven approach to stochastic search in machine learning.

MILP [8] is an ILP algorithm that performs stochastic search for single clauses to overcome the myopic behavior of greedy search. The outer loop of the algorithm employs the more conventional separate-and-conquer strategy.

SUBDUE [4] is an MDL-based algorithm for substructure discovery in graphs. It differs from SP in that its search is class-blind, and that it basically performs a beam search (the current clauses are the parent clauses of the next generation).

5 Conclusion and Further Work

In this paper we presented a stochastic approach to propositionalization of relational background knowledge. Our next steps will be to perform additional experiments in several other domains. Secondly, we will experiment with a Bayesian evaluation of clauses (by their posterior), so that there will be a penalty for overly complex clauses. Thirdly, we are planning to complement the type of search presented here by class-blind search.

Acknowledgements

This research is sponsored by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grant number P10489-MAT. Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian Federal Ministry of Science and Transport. We would like to thank Ross King and Ashwin Srinivasan for providing the carcinogenicity data, and Gerhard Widmer for valuable discussions.

References

[1] P. Auer, W. Maass, and R. Holte. Theory and applications of agnostic pac-learning with small decision trees. In A. Prieditis and S. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning (ML95)*. Morgan Kaufmann, 1995.

[2] W.W. Cohen. Pac-learning nondeterminate clauses. In *Proc. Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1994.

[3] W.W. Cohen. Learning trees and rules with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 709–716, 1996.

[4] D.J. Cook and L.B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[5] P. Geibel and F. Wysotzki. Relational learning with decision trees. In *Proc. Twelfth European Conference on Artificial Intelligence (ECAI-96)*, pages 428–432, 1996.

[6] A. Giordana, L. Saitta, and F. Zini. Learning disjunctive concepts by means of genetic algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 96–104, 1994.

[7] R.D. King and A. Srinivasan. Prediction of rodent carcinogenicity bioassays from molecular structure using inductive logic programming. *Environmental Health Perspectives*, 1997.

[8] M. Kovačič. MILP: a stochastic approach to Inductive Logic Programming. In *Proceedings of the Fourth International Workshop on Inductive Logic Programming (ILP-94)*, GMD-Studien Nr. 237, pages 123–138, 1994.

[9] S. Kramer. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996.

[10] N. Lavrac and S. Džeroski. *Inductive Logic Programming*. Ellis Horwood, Chichester, UK, 1994.

[11] S. Muggleton. Inverse Entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

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

[13] J.R. Quinlan. Learning with continuous classes. In Sterling Adams, editor, *Proceedings AI'92*, pages 343–348, Singapore, 1992. World Scientific.

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

[15] G. Silverstein and M.J. Pazzani. Relational clichés: Constraining constructive induction during relational learning. In L.A. Birnbaum and G.C. Collins, editors, *Machine Learning: Proceedings of the Eighth International Workshop (ML91)*, pages 203–207, San Mateo, CA, 1991. Morgan Kaufmann.

[16] A. Srinivasan and R.D. King. Feature construction with Inductive Logic Programming: a study of quantitative predictions of chemical activity aided by structural attributes. In *Proceedings of the 6th International Workshop on Inductive Logic Programming (ILP-96)*, 1996.

[17] P. Turney. Low size-complexity Inductive Logic Programming: the East-West challenge considered as a problem in cost-sensitive classification. In *Proceedings of the 5th International Workshop on Inductive Logic Programming (ILP-95)*, pages 247–263. Katholieke Universiteit Leuven, 1995.

[18] J.D. Zucker and J.G. Ganascia. Representation changes for efficient learning in structural domains. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 543–551, 1996.