kFOIL: Learning Simple Relational Kernels

Niels Landwehr[1] and **Andrea Passerini**[2] and **Luc De Raedt**[1] and **Paolo Frasconi**[2]

[1]Machine Learning Lab
Department of Computer Science
Albert-Ludwigs Universität, Freiburg, Germany
{landwehr,deraedt}@informatik.uni-freiburg.de

[2]Machine Learning and Neural Networks Group
Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze, Florence, Italy
{passerini,p-f}@dsi.unifi.it

Abstract

A novel and simple combination of inductive logic programming with kernel methods is presented. The kFOIL algorithm integrates the well-known inductive logic programming system FOIL with kernel methods. The feature space is constructed by leveraging FOIL search for a set of relevant clauses. The search is driven by the performance obtained by a support vector machine based on the resulting kernel. In this way, kFOIL implements a *dynamic* propositionalization approach. Both classification and regression tasks can be naturally handled. Experiments in applying kFOIL to well-known benchmarks in chemoinformatics show the promise of the approach.

Introduction

Various successes have been reported in applying inductive logic programming (ILP) techniques to challenging problems in bio- and chemoinformatics, cf. e.g. (Bratko & Muggleton 1995). These successes can—to a large extent—be explained by the use of an expressive general purpose representation formalism that allows one to deal with structured data, to incorporate background knowledge in the learning process, and to obtain hypotheses in the form of a small set of rules that are easy to interpret by domain experts.

On the other hand, support vector machines and kernel methods in general have revolutionized the theory and practice of machine learning in the past decade. These methods do not only yield highly accurate hypotheses; they are also grounded in a solid mathematical theory. However, dealing with structured data and employing background knowledge is harder, as it typically requires one to develop a novel kernel for the specific problem at hand, which is a non-trivial task. Also, the resulting hypotheses are hard to interpret by the human domain expert.

Given these developments, it can be no surprise that several researchers have started to combine and integrate ideas from ILP with those from support vector machines. First, there has been a significant interest in developing kernels for structured data, cf. (Gaertner 2003) for an overview, in particular for sequences, trees, graphs, and even individuals described in high-order logic (Gaertner, Lloyd, & Flach 2004). All these kernels are fixed *before* learning takes place and, to the best of the authors' knowledge, a kernel method that directly learns from relational representations is still missing. Second, there is the idea of *static* propositionalization, in which an ILP problem is turned into a propositional one by pre-computing a typically large set of features, cf. e.g. (Muggleton, Amini, & Sternberg 2005), and then using traditional SVM learning on the resulting representation. An extension of this approach transforms the relational representations into a structured one, by e.g. computing proof-trees for so-called visitor programs (Passerini, Frasconi, & De Raedt 2006). Third, as kernels are closely related to similarity measures, work on distance based relational learning (Ramon & Bruynooghe 1998; Kirsten, Wrobel, & Horváth 2001) should also be mentioned. The drawback of these approaches is that the resulting models are still complex and hard to interpret. In addition, the user typically needs to specify additional information to restrict the number of features generated in the propositionalization process or to encode the distance function, which is often a non-trivial task.

The approach taken in this paper is different. The key idea is to *dynamically* induce a small set of clauses using a FOIL-like covering algorithm (Quinlan 1990) and to use these as features in standard kernel methods. Applying rule-learning principles leads to a typically small set of rules or features, which are—due to the use of a relational representation—also easy to interpret. Using these features to define a kernel leads to similarity measures amongst relational examples and also allows to directly tackle a wide variety of learning tasks including classification and regression with support vector machines. Especially the uniform treatment of classification and regression is appealing from an ILP perspective, as these typically require rather different techniques (with possibly the exception of decision trees (Kramer 1996)).In contrast to the three types of approaches mentioned earlier, the kernel or similarity measure is being learned. Also, whereas the resulting model is still a kind of propositionalization, the features are learned dynamically and not pre-computed in advance. Thus a *dynamic* propositionalization technique results, which is similar in spirit to the nFOIL system (Landwehr, Kersting, & De Raedt 2005), a method that combines FOIL with naïve Bayes and proved to yield significant improvements over traditional ILP methods such as Aleph (an ILP system developed by Ashwin

Srinivasan [1]) on a number of benchmark problems.

The above sketched idea has been incorporated in the kFOIL algorithm and has been elaborated for classification as well as regression problems. kFOIL has been evaluated experimentally on a number of well-known benchmark problems from the field of ILP.

Problem Specification

We start from an inductive logic programming perspective and then extend it towards the use of kernels.

Inductive Logic Programming

Traditional ILP approaches tackle the following problem:

Given

- a background theory B, in the form of a set of definite clauses, i.e., clauses of the form $h \leftarrow b_1, \cdots, b_k$ where h and the b_i are logical atoms;
- a set of examples E in the form of ground facts of an unknown target function y; y maps examples to $\{+1, -1\}$ (denoting $\{true, false\}$) in a classification setting, or alternatively to $\mathbb{R}$, the reals, in a regression setting;
- a language of clauses $\mathcal{L}$, which specifies the clauses that are allowed in hypotheses;
- a $f(e, H, B)$ function, which returns the value of the hypothesis H on the example e w.r.t. the background theory B;
- a $score(E, H, B)$ function, which specifies the quality of the hypothesis H w.r.t. the data E and the background theory;

Find $\arg\max_{H \subset \mathcal{L}} score(E, H, B)$.

In a *classification* setting, the goal typically is to find a complete and consistent concept-description, i.e., a set of clauses that cover all positive and no negative examples. This can be formalized within our framework by making the following choices for $f(e, H, B)$ and $score$:

- $f(e, H, B)$ = +1 if $B \cup H \models e$ (i.e., e is entailed by $B \cup H$); otherwise, $f(e, H, B)$ = -1;
- $score(E, H, B)$ = training set accuracy.

In a *regression* setting, the goal is typically to find a hypothesis H that minimizes a measure such as the root mean squared error between the target $y(e)$ and the prediction $f(e, H, B)$. This can be modeled by setting $score(E, H, B) = -RMSE(E, H, B)$.

kFOIL's Problem Specification

Let us now show how kFOIL can be formulated within the above sketched definition of inductive logic programming. The notions of examples, language, hypotheses and background theory remain essentially the same. However, it is extended by a notion of similarity between pairs of examples e_1,e_2 that is defined—as for other kernel methods—by a kernel function. ¿From an ILP point of view, this should take into account the hypothesis H and the background theory B. Thus kFOIL requires a kernel K of the form $K(e_1, e_2, H, B)$. As the background theory B is fixed throughout the whole learning process, we will from now on omit this argument from the notation. The function K plays a role similar to that of the distances between first-order logic objects used in relational learning (Ramon & Bruynooghe 1998; Kirsten, Wrobel, & Horváth 2001). A support vector machine will then be used in combination with the kernel K to define the $f(e, H, B)$ function.

Kernel functions based on clauses To obtain kernels $K(e_1, e_2, H)$, it is convenient to first propositionalize the examples e_1 and e_2 using H and B and then to employ existing kernels on the resulting problem. The natural way of doing this, is to map each example e onto a vector $\varphi_H(e)$ over $\{0, 1\}^n$ with $n = |H|$, having $\varphi_H(e)_i = 1$ if $B \cup \{c_i\} \models e$ for the i-th clause $c_i \in H$, and 0 otherwise.

Example 1 *Consider the following background theory B, which describes the structure of molecules:*

$$\begin{array}{ll}
atm(m1, a1\_1, c, 22, -0.11) & bond(m1, a1\_1, a1\_2, 7) \\
atm(m1, a1\_3, c, 22, 0.02) & bond(m1, a1\_3, a1\_4, 7) \\
atm(m1, a1\_26, o, 40, -0.38) & bond(m1, a1\_18, a1\_26, 2) \\
\ldots & \ldots \\
atm(m2, a2\_1, c, 22, -0.11) & bond(m2, a2\_1, a2\_2, 7) \\
atm(m2, a2\_3, c, 27, 0.02) & bond(m2, a2\_3, a2\_4, 2) \\
atm(m2, a2\_26, o, 40, -0.38) & bond(m2, a2\_18, a2\_26, 7)
\end{array}$$

and the examples $m1, m2$. A possible hypothesis $H = \{c_1, c_2, c_3\}$ for this domain is

$$\begin{array}{l}
pos(X) \leftarrow atm(X, A, o, 40, C) \\
pos(X) \leftarrow atm(X, A, c, 22, C), atm(X, B, E, 22, 0.02) \\
pos(X) \leftarrow atm(X, A, c, 27, C), bond(X, A, B, 2)
\end{array}$$

H as a logical theory covers both examples. Clauses c_1, c_2 succeed on the first example and clauses c_1, c_3 on the second. Consequently, in the feature space spanned by the truth values of the clauses, the examples are represented as

$$\varphi_H(m1) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \quad \varphi_H(m2) = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Let us now look at the effect of defining kernels on the propositionalized representation. A simple linear kernel K_L would give the following results:

$$\begin{array}{rcl}
K_L(m1, m2, H) & = & \langle \varphi_H(m1), \varphi_H(m2) \rangle = 1 \\
K_L(m1, m1, H) & = & \langle \varphi_H(m1), \varphi_H(m1) \rangle = 2 \\
K_L(m2, m2, H) & = & \langle \varphi_H(m1), \varphi_H(m1) \rangle = 2
\end{array}$$

The resulting kernel K_L can be interpreted as the number of clauses in H that succeed on both examples.

Let us formalize the linear kernel introduced in the above example in terms of logical entailment:

$$K_L(e_1, e_2, H) = \#ent_H(e_1 \wedge e_2)$$

where $\#ent_H(f) = |\{c \in H | B \wedge \{c\} \models f\}|$ denotes the number of clauses in H that together with B logically entail

[1] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html

f. Intuitively, this implies that two examples are similar if they share many structural features. Which structural features to look at when computing similarities is encoded in the hypothesis H.

This formalism can be generalized to standard polynomial (K_P) and Gaussian (K_G) kernels. Using a polynomial kernel, the interpretation in terms of logical entailment is

$$K_P(e_1, e_2, H) = (\#ent_H(e_1 \wedge e_2) + 1)^p,$$

which amounts to considering conjunctions of up to p clauses which logically entail the two examples, as can easily be shown by explicitly computing the feature space induced by the kernel. Using a Gaussian kernel turns out to implement the similarity

$$K_G(e_1, e_2, H) = exp\left(-\frac{\#ent_H((e_1 \vee e_2) \wedge \neg(e_1 \wedge e_2))}{2\sigma^2}\right)$$

where the argument of ent_H can be interpreted as a kind of symmetric difference between the two examples.

From similarity to classification Having defined kernels over examples in a propositional representation, we only need to employ them within traditional support vector machine methods to obtain effective classification and regression algorithms.

For instance, using the standard support vector method for classification, the $f(e, H, B)$ function is expressed as

$$f(e, H, B) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y(e_i) K(e, e_i, H) + b\right) \quad (1)$$

where $\{e_1, ..., e_m\}$ are the training examples and $y(e_i) = 1$ if e_i is a positive example and $y(e_i) = -1$ otherwise. Similarly, using support vector regression one obtains

$$f(e, H, B) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(e, e_i, H) + b. \quad (2)$$

The support vector coefficients α_i,α_i^* and the bias b can be obtained from the theory H using standard support vector training.

By now, we have formally specified the learning setting addressed by kFOIL. It is the instantiation of the standard ILP problem sketched earlier with the $f(e, H, B)$ function just defined. As scoring functions, kFOIL employs training set accuracy for classification and Pearson correlation or root mean squared error for regression. The key point is that kFOIL—as standard inductive logic programming techniques—must find the right hypothesis H that maximizes its score. Note that this approach differs significantly from the static propositionalization approaches, where H is actually pre-computed and fixed. As kFOIL learns the hypothesis H, this implies that the kernel itself is being learned.

The kFOIL Learning Algorithm

To learn H, kFOIL employs an adaptation of the well-known FOIL algorithm (Quinlan 1990), which essentially implements a *separate-and-conquer* rule learning algorithm in a relational setting.

Algorithm 1 Generic FOIL algorithm.

```
Initialize $H := \emptyset$
repeat
  Initialize $c := p(X_1, \cdots, X_n) \leftarrow$
  repeat
    for all $c' \in \rho(c)$ do
      compute $score(E, H \cup \{c'\}, B)$
    end for
    let $c$ be the $c' \in \rho(c)$ with the best score
  until stopping criterion
  add $c$ to $H$
  $E := update(E, H)$
until stopping criterion
output $H$
```

The generic FOIL algorithm is sketched in Algorithm 1. It repeatedly searches for clauses that score well with respect to the data set and the current hypothesis and adds them to the current hypothesis. The examples covered by a learned clause are removed from the training data (in the *update* function). In the inner loop, it greedily searches for a clause that scores well. To this aim, it employs a general-to-specific hill-climbing search strategy. Let $p(X_1, ..., X_n)$ denote the predicate that is being learned (e.g, pos(X) for a simple classification problem). Then the most general clause, which succeeds on all examples, is "$p(X_1, ..., X_n) \leftarrow$". The set of all refinements of a clause c within the language bias is produced by a *refinement operator* $\rho(c)$. For our purposes, a refinement operator just specializes a clause $h \leftarrow b_1, \cdots, b_k$ by adding a new literal b_{k+1}, though other refinements have also been used in the literature. This type of algorithm has been successfully applied to a wide variety of problems in ILP. Many different scoring functions and stopping criteria have been employed.

The search in kFOIL follows the generic search strategy outlined in Algorithm 1. However, there are three key differences, which will now be outlined. First, when scoring a refined clause, a support vector machine based on the current kernel including the clause has to be built and its performance must be evaluated on the training data. This can be achieved by introducing a loss function $V(y(e), f(e))$ that measures the cost of predicting $f(e)$ when the target is $y(e)$. Thus $score(E, H \cup \{c'\}, B)$ is computed in a "wrapper" fashion as follows:

```
$(\alpha_1, ..., \alpha_m, b) := train\_svm(E, H \cup \{c'\}, B)$
for all $e \in E$ do
  compute $V(y(e), f(e, H \cup \{c'\}, B))$
end for
output $score(E, H \cup \{c'\}, B)$
```

Here $train\_svm(E, H, B)$ trains a support vector machine using the kernel defined by H, while $f(e, H, B)$ computes the prediction according to Equation 1 or Equation 2 for the classification or regression case respectively.

Second, kFOIL cannot use a separate-and-conquer approach. Because the final model in FOIL is the logical disjunction of the learned clauses, (positive) examples that are already covered by a learned clause can be removed from the training data (in the $update(E, H)$ function in Algorithm 1). In kFOIL, this notion of coverage is lost, and the training set is not changed between iterations. Therefore, $update(E, H)$ returns E. Finally, FOIL stops when it fails to find a clause that covers additional positive examples. As an equally simple stopping criterion, learning in kFOIL is stopped when the improvement in score between two successive iterations falls below a certain threshold.

The repeated support vector optimizations performed during the search are computationally expensive. However, the costs can be reduced with simple tabling techniques, and by exploiting the fact that the relational example space is mapped to a much simpler propositional space by φ_h. There, different relational examples are represented by the same vector, and can be merged to one example with a higher weight. In our experimental study, this typically reduced the time needed to learn a model by one to two orders of magnitude.

In a preliminary evaluation, we compared alternative scores to guide FOIL search, including kernel target alignment (Lanckriet *et al.* 2004) and various loss functions V in the wrapper-style score algorithm above (hinge loss, 0-1 loss, margin-based conditional likelihood). Kernel target alignment does not require SVM training but the speedup is marginal due to the inherent cost of FOIL and the optimizations outlined above. In addition, local optima problems occurred in conjunction with greedy search. 0-1 loss for classification and quadratic loss for regression yielded the most stable search results and were employed in the experiments reported below. These criteria are known to be associated with the risk of overfitting in the case of propositional feature selection (Kohavi & John 1997). However, the use of independent data—e.g. by using a leave-one-out estimated loss as suggested in (Reunanen 2003)—would increase complexity significantly and the more efficient approach of estimating leave-one-out bounds resulted in unstable search.

Experimental Evaluation

Several questions arise w.r.t. the kernel based dynamic propositionalization approach developed in kFOIL:

(Q1) Is kFOIL competitive with state-of-the-art inductive logic programming systems for classification?

(Q2) Is kFOIL competitive with state-of-the-art inductive logic programming systems for regression?

(Q3) Is kFOIL competitive with other dynamic propositionalization approaches, in particular to nFOIL?

(Q4) Is kFOIL competitive with static propositionalization approaches?

Datasets and Algorithms

We conducted experiments on nine benchmark datasets from four domains. On **Mutagenesis** (Srinivasan *et al.* 1996) the problem is to predict the mutagenicity of a set of compounds We used atom and bond information only. For **Alzheimer** (King, Srinivasan, & Sternberg 1995), the aim is to compare four desirable properties of drugs against Alzheimer's disease: inhibit **amine** reuptake (686 examples), low **toxicity** (886 examples), high **acetyl** cholinesterase inhibition (1326 examples), and good **reversal** of memory deficiency (642 examples).

The **NCTRER** dataset has been extracted from the EPA's DSSTox NCTRER Database (Fang *et al.* 2001). It contains structural information about a diverse set of 232 natural, synthetic and environmental estrogens and classifications with regard to their binding activity for the estrogen receptor. Again, we used atom and bond information only. In the **Biodegradability** domain (Blockeel *et al.* 2004) the task is to predict the biodegradability of 328 chemical compounds based on their molecular structure and global molecular measurements. This is originally a regression task, but can also be transformed into a classification task by putting a threshold on the target variable.

On Mutagenesis, Alzheimer, and NCTRER, kFOIL was compared to nFOIL, the state-of-the-art ILP system Aleph and a static propositionalization approach. We used a variant of the relational frequent query miner WARMR (Dehaspe, Toivonen, & King 1998) for static propositionalization as WARMR patterns have shown to be effective propositionalization techniques on similar benchmarks in inductive logic programming (Ashwin Srinivasan 1999). The variant used was c-ARMR (De Raedt & Ramon 2004), which allows to remove redundancies amongst the found patterns by focusing on so-called free patterns. c-ARMR was used to generate all free frequent patterns in the data sets where the frequency threshold was set to 20%. We used at most 5000 of the generated patterns as features to generate (binary) propositional representations of the datasets. On the propositionalized datasets, a cross-validation of a support vector machine was then performed[2]. To evaluate the regression performance of kFOIL, we reproduced the experimental setting used in (Blockeel *et al.* 2004) and compared to the results obtained in that study for Tilde and S-CART.

As the goal of the experimental study was to verify that the presented approach is competitive to other state-of-the-art techniques, and not to boost performance, we did not try to specifically optimize any parameter. For nFOIL, we used the default settings: maximum number of clauses in a hypothesis was set to 25, maximum number of literals in a clause to 10 and the threshold for the stopping criterion to 0.1%. For kFOIL, we used exactly the same parameters. For both algorithms, a beam search with beam size 5 instead of simple greedy search was performed, as in (Landwehr, Kersting, & De Raedt 2005). Furthermore, a polynomial kernel of degree 2 was used, the regularization constant C was set to 1 for classification and 0.01 for regression, and the ϵ tube parameter was set to 0.001. All SVM parameters were set identical for all datasets, and kept fixed during the search for clauses.

Results

Table 1 shows cross-validated predictive accuracy results on Mutagenesis, Alzheimer, and NCTRER. Both kFOIL and nFOIL on average yield higher predictive accuracies

[2]Note that this methodology puts this approach at a slight advantage and might yield over-optimistic results.

Dataset	kFOIL	nFOIL	Aleph	c-ARMR+SVM
Mutagenesis r.f.	81.3 ± 11.0	75.4 ± 12.3	73.4 ± 11.8	73.9 ± 11.2
Mutagenesis r.u.	81.0 ± 40.0	78.6 ± 41.5	85.7 ± 35.4	76.2 ± 43.1
Alzheimer amine	88.8 ± 5.0	86.3 ± 4.3	70.2 ± 7.3●	81.2 ± 4.5●
Alzheimer toxic	89.3 ± 3.5	89.2 ± 3.4	90.9 ± 3.5	71.6 ± 1.9●
Alzheimer acetyl	87.8 ± 4.2	81.2 ± 5.2●	73.5 ± 4.3●	72.4 ± 3.6●
Alzheimer memory	80.2 ± 4.0	72.9 ± 4.3●	69.3 ± 3.9●	68.7 ± 3.0●
NCTRER	77.6 ± 9.4	78.0 ± 9.1	50.9 ± 5.9●	65.1 ± 13.2●

Table 1: Average predictive accuracy results on Mutagenesis, Alzheimer and NCTRER for kFOIL, nFOIL, Aleph and static propositionalization. On Mutagenesis r.u. a leave-one-out cross-validation was used (which, combined with the small size of the dataset, explains the high variance of the results), on all other datasets a 10 fold cross-validation. ● indicates that the result for kFOIL is significantly better than for other method (paired two-sided t-test, p = 0.05).

Dataset	kFOIL	Tilde	S-CART
Classification			
BioDeg Global + R	74.3 ± 0.76	73.6 ± 1.1	72.6 ± 1.1●
BioDeg Global + P1 + P2 + R	73.2 ± 2.0	72.9 ± 1.1	71.3 ± 2.3
Regression: correlation			
BioDeg Global + R	0.609 ± 0.047	0.616 ± 0.021	0.605 ± 0.023
BioDeg Global + P1 + P2 + R	0.597 ± 0.026	0.595 ± 0.020	0.606 ± 0.032
Regression: root mean squared error			
BioDeg Global + R	1.196 ± 0.023	1.265 ± 0.033●	1.290 ± 0.038●
BioDeg Global + P1 + P2 + R	1.290 ± 0.037	1.335 ± 0.036	1.301 ± 0.049

Table 2: Result on the Biodegradability dataset. The results for Tilde and S-CART have been taken from (Blockeel *et al.* 2004). 5 runs of 10 fold cross-validation have been performed, on the same splits into training and test set as used in (Blockeel *et al.* 2004). For classification, average accuracy is reported, for regression, Pearson correlation and RMSE. ● indicates that the result for kFOIL is significantly better than for other method (unpaired two-sided t-test, p = 0.05).

than the ILP system Aleph and static propositionalization. kFOIL significantly outperforms nFOIL on two datasets, and a Wilcoxon Matched Pairs Test applied to the results of kFOIL and nFOIL on the different datasets shows that kFOIL reaches significantly higher predictive accuracy on average (p=0.05). These results affirmatively answer questions **Q1**–**Q3**.

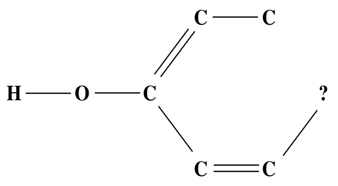
Table 2 shows results for the Biodegradability dataset. For regression, we ran kFOIL with scoring based on correlation and root mean squared error, and measured the result using the corresponding evaluation criterion. The results obtained show that kFOIL is competitive with the first-order decision tree systems S-CART and Tilde for classification. For regression, it is competitive at maximizing correlation, and slightly superior at minimizing RMSE. Thus, question **Q4** can be answered affirmatively as well.

kFOIL returned between 2.8 and 22.9 clauses averaged over the folds of the cross-validation, depending on the dataset. Interestingly, the number of clauses in H was always lower than for nFOIL. On the datasets we examined, building a kFOIL model takes up to 10 minutes for classification, and up to 30 minutes for regression. This is of the same order of magnitude as the runtime for the other systems considered.

Finally, we give an example of a learned clause which is meaningful to human domain experts: on the NCTRER dataset, we obtained the clause

$$\begin{gathered}\leftarrow atm(B,o), bd\_atm(B,C,c,-), bd\_atm(C,D,c,=),\\ bd\_atm(C,E,c,-), bd\_atm(E,F,c,=),\\ bd\_atm(G,D,c,-), bd\_atm(F,H,I,-).\end{gathered}$$

It encodes an aromatic ring with a phenol group (a so-called phenolic ring):



In the study presented in (Fang *et al.* 2001), the presence of a phenolic ring is identified by human experts as one of the main factors that determine estrogen-binding activity of chemical compounds.

Related Work and Conclusions

We have presented the kFOIL system, which introduces a simple integration of inductive logic programming methods with support vector learning. kFOIL can be considered a propositionalization approach. Two types of propositionalization approaches have been discussed: static ones, in which a typically large set of features is pre-computed, and dynamic propositionalization, in which features are incrementally and greedily generated. As the generation of clauses is driven by the performance of the support vector machine, kFOIL performs dynamic propositionalization. Hence, kFOIL is related to Support Vector Inductive Logic Programming,which combines static propositionalization with support vector learning, and systems like SAYU (Davis *et al.* 2005), nFOIL, and Structural Logistic Regression (Popescul *et al.* 2003), which all combine dynamic propositionalization with probabilistic models. In contrast, kFOIL employs kernel based learning, which allows to tackle classification and regression problems in a uniform framework. Also, kFOIL improved upon nFOIL in terms of predictive accuracy in our experimental study.

From a kernel machine perspective, kFOIL can also be seen as constructing the kernel based on the available data and therefore it has interesting connections to methods that attempt to learn the kernel from data. The method by (Lanckriet *et al.* 2004) works in the transductive setting (input portion of the test data available when training) and uses a semidefinite programming algorithm for computing the optimal kernel matrix. Algorithms for learning the kernel function include the idea of using a hyperkernel (that spans a Hilbert space of kernel functions) (Ong, Smola, & Williamson 2002) and the use of regularization functionals (Micchelli & Pontil 2005). These approaches are typically more principled than kFOIL (as they learn the kernel by solving well-posed optimization problems). However the formulation by which the kernel is obtained as a convex combination of other kernel functions would be difficult or impossible to apply in the context of dynamic feature construction in a fully-fledged relational setting. Furthermore, to the best of the authors' knowledge, no other method proposed so far can learn kernels defined by small sets of interpretable first-order rules.

Acknowledgements The authors would like to thank Kristian Kersting and the anonymous reviewers for valuable comments. The research was supported by the European Union IST programme, contract no. FP6-508861, *Application of Probabilistic Inductive Logic Programming II.*

References

Ashwin Srinivasan, Ross D. King, D. B. 1999. An Assessment of ILP-Assisted Models for Toxicology and the PTE-3 Experiment. In *Proc. of ILP'99*.

Blockeel, H.; Dzeroski, S.; Kompare, B.; Kramer, S.; Pfahringer, B.; and Laer, W. 2004. Experiments in Predicting Biodegradability. *Appl. Art. Int.* 18(2):157–181.

Bratko, I., and Muggleton, S. 1995. Applications of Inductive Logic Programming. *Comm. of the ACM* 38(11):65–70.

Davis, J.; Burnside, E.; de Castro Dutra, I.; Page, D.; and Costa, V. S. 2005. An Integrated Approach to Learning Bayesian Networks of Rules. In *Proc. of ECML'05*, 84–95.

De Raedt, L., and Ramon, J. 2004. Condensed Representations for Inductive Logic Programming. In *Proc. of KR'04*.

Dehaspe, L.; Toivonen, H.; and King, R. 1998. Finding Frequent Substructures in Chemical Compounds. In *Proc. of KDD'98*.

Fang, H.; Tong, W.; Shi, L.; Blair, R.; Perkins, R.; Branham, W.; Hass, B.; Xie, Q.; Dial, S.; Moland, C.; and Sheehan, D. 2001. Structure-Activity Relationships for a Large Diverse Set of Natural, Synthetic, and Environmental Estrogens. *Chemical Research in Toxicology* 14(3):280–294.

Gaertner, T.; Lloyd, J.; and Flach, P. 2004. Kernels and Distances for Structured Data. *Machine Learning* 57(3):205–232.

Gaertner, T. 2003. A Survey of Kernels for Structured Data. *SIGKDD Explorations* 5(1):49–58.

King, R.; Srinivasan, A.; and Sternberg, M. 1995. Relating Chemical Activity to Structure: an Examination of ILP Successes. *New Generation Computing* 13(2,4):411–433.

Kirsten, M.; Wrobel, S.; and Horváth, T. 2001. Distance based approaches to relational learning and clustering. In *Relational Data Mining*, 213–230. Springer.

Kohavi, R., and John, G. 1997. Wrappers for feature subset selection. *Art. Int.* 97(1–2):273–324.

Kramer, S. 1996. Structural Regression Trees. In *Proc. of AAAI*, 812–819.

Lanckriet, G. R. G.; Cristianini, N.; Bartlett, P.; Ghaoui, L. E.; and Jordan, M. I. 2004. Learning the Kernel Matrix with Semidefinite Programming. *J. Mach. Learn. Res.* 5:27–72.

Landwehr, N.; Kersting, K.; and De Raedt, L. 2005. nFOIL: Integrating Naïve Bayes and FOIL. In *Proc. of AAAI*, 795–800.

Micchelli, C. A., and Pontil, M. 2005. Learning the Kernel Function via Regularization. *J. Mach. Learn. Res.* 6:1099–1125.

Muggleton, S.; Amini, A.; and Sternberg, M. 2005. Support Vector Inductive Logic Programming. In *Proc. of DS'05*, 163–175.

Ong, C. S.; Smola, A. J.; and Williamson, R. C. 2002. Hyperkernels. In *NIPS 15*.

Passerini, A.; Frasconi, P.; and De Raedt, L. 2006. Kernels on prolog proof trees: Statistical learning in the ILP setting. *J. Mach. Learn. Res.* 7:307–342.

Popescul, A.; Ungar, L.; Lawrence, S.; and Pennock, D. 2003. Statistical Relational Learning for Document Mining. In *Proc. of ICDM'03*, 275–282.

Quinlan, J. 1990. Learning Logical Definitions from Relations. *Machine Learning* 5:239–266.

Ramon, J., and Bruynooghe, M. 1998. A Framework for Defining Distances Between First-Order Logic Objects. In *Proc. of ILP*, 271–280.

Reunanen, J. 2003. Overfitting in making comparisons between variable selection methods. *J. Mach. Learn. Res.* 3:1371–1382.

Srinivasan, A.; Muggleton, S.; King, R.; and Sternberg, M. 1996. Theories for Mutagenicity: a Study of First-Order and Feature-Based Induction. *Art. Int.* 85:277–299.