Mach Learn (2010) 78: 305–342
DOI 10.1007/s10994-009-5163-1

Fast learning of relational kernels

Niels Landwehr · Andrea Passerini · Luc De Raedt · Paolo Frasconi

Received: 27 October 2008 / Revised: 24 August 2009 / Accepted: 15 November 2009 /
Published online: 5 January 2010

Abstract We develop a general theoretical framework for statistical logical learning with kernels based on dynamic propositionalization, where structure learning corresponds to inferring a suitable kernel on logical objects, and parameter learning corresponds to function learning in the resulting reproducing kernel Hilbert space. In particular, we study the case where structure learning is performed by a simple FOIL-like algorithm, and propose alternative scoring functions for guiding the search process. We present an empirical evaluation on several data sets in the single-task as well as in the multi-task setting.

Keywords Statistical relational learning · Inductive logic programming · Kernel methods · Dynamic propositionalization · Kernel learning · Multi-task learning

Editor: Lise Getoor.

N. Landwehr (✉)
Department for Computer Science, University of Potsdam, August-Bebel-Str. 89, 14482 Potsdam, Germany
e-mail: landwehr@cs.uni-potsdam.de

A. Passerini
Dipartimento di Ingegneria e Scienza dell'Informazione, Università degli Studi di Trento, Via Sommarive 14, 38100 Povo (TN), Italy
e-mail: passerini@disi.unitn.it

L. De Raedt
Departement Computerwetenschappen, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
e-mail: luc.deraedt@cs.kuleuven.be

P. Frasconi
Machine Learning and Neural Networks Group, Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Via di Santa Marta 3, 50139 Firenze, Italy
e-mail: p-f@dsi.unifi.it

1 Introduction

Inductive logic programming research has focused on concept-learning from examples. Within the field of machine learning, ILP has stressed the need for producing hypotheses that can be readily interpreted by human experts and that can be used for providing explanations for predictions. This explains why traditional ILP methods have not always incorporated statistical robustness principles needed to handle noise. The recently emerged field of probabilistic ILP, also known as statistical relational learning (SRL) (Getoor and Taskar 2007; De Raedt et al. 2008) attempts to alleviate this situation while retaining the ability to explain the data. Most efforts in SRL have focused on extensions of various probabilistic graphical models, and lifting them to first-order logic. These approaches typically separate the structure learning aspect, which aims at identifying the underlying logical description of the data, from the parameter learning, by which adjustable numerical values are attached to logical symbols (e.g., ground clauses). Although this is a very natural and promising direction, the efficiency of inference and structure learning procedures remains a major issue.

Another dominating paradigm in current machine learning research is based on learning linear functions in a suitable reproducing kernel Hilbert space (RKHS), mostly by means of convex optimization procedures. Undoubtedly, this setting has offered remarkable advantages, including the simplicity due to the uniqueness of the solution, the efficiency of well understood optimization procedures, the ability to control overfitting by means of regularization, and the flexibility of abstracting away the type of the data points via implicit feature mappings. This well established framework, unfortunately, also comes with some limitations. One is that the kernel function needs to be carefully designed for the problem at hand. Significant efforts have been devoted to devising effective kernels for specialized data types, especially structured or relational types (Gärtner 2003; Gärtner et al. 2004; Leslie et al. 2002; Lodhi et al. 2002). Designing the right kernel, however, requires a sufficient understanding of the application domain so that one can figure out the relevant and important features for the problem at hand. While it is possible to define "universal" kernels defining arbitrarily complex hypothesis spaces (Micchelli et al. 2006; Caponnetto et al. 2008), this may seriously affect the generalization and representation efficiency. Undesired effects have been described in many ways, such as diagonal dominance (Weston et al. 2003), hardness of finding a large margin in the feature space (Ben-David et al. 2002), or inefficiency in terms of the number of examples needed to represent highly varying functions (Bengio et al. 2005). Ideally, the kernel function, or the set of relevant features, should be learned from data, rather than designed by hand. Research on kernel learning has mainly focused on methods for combining existing base kernels with appropriate weights (Argyriou et al. 2007; Lanckriet et al. 2004; Ong et al. 2005; Micchelli and Pontil 2005). Base kernels can be manually designed by domain experts or represent families of continuously parameterized kernels such as Gaussians with varying covariance (Argyriou et al. 2006). However, little research exists in learning kernels from scratch in a fully relational domain. Learning the kernel function arguably stands at a higher level of abstraction, as compared to function learning in a given RKHS. The relationship between kernel learning and function learning actually shares some similarities with the relationship between structure and parameter learning in probabilistic ILP and statistical relational learning. This suggests that discrete-space search algorithms (such as those employed in structure learning for probabilistic ILP) may also be useful for inducing kernel functions in a logical setting.

Following this suggestion, this paper explores a combination of ILP and kernel methods building upon the kFOIL algorithm initially presented in Landwehr et al. (2006). More

specifically, we use ILP as a form of structure learning for inducing a suitable kernel function from data in a *dynamic propositionalization* framework (Landwehr et al. 2005). Dynamic propositionalization means that the feature set and the kernel machine are optimized jointly; this is in contrast to static propositionalization techniques that select a feature set a priori to propositionalize the data and afterwards apply a statistical classifier. We show that this approach is practically viable and that our most recent versions of kFOIL are efficient enough to handle data sets of moderately large size. We also show that the approach can be easily adapted to perform multi-task learning (Caruana 1997), a setting that has received significant attention within the statistical learning community but has only rarely been addressed within the ILP setting. Interestingly, the multi-task version of kFOIL is advantageous both in terms of accuracy and efficiency.

The rest of the paper is organized as follows: in Sect. 2 we give some background on ILP and kernel machines for statistical learning; in Sect. 3 we introduce our framework for statistical relational learning with kernels, and present kFOIL as a simple instantiation within such framework; Sect. 4 discusses related work; and Sect. 5 presents some efficiency-related algorithmic details. Finally, an extensive experimental evaluation is reported in Sect. 6, and some conclusions are drawn in Sect. 7.

2 Background

2.1 Inductive logic programming

Traditional inductive logic programming addresses the task of inducing a concept which explains a set of examples. Logic is used to represent both examples and concepts to be learned (for an introduction to logic see, for example, Lloyd 1987 and for ILP, De Raedt 2008). More specifically, definite clauses, which form the basis for the programming language Prolog, are used to represent examples and concepts. A *definite clause* is an expression of the form $h \leftarrow b_1, \ldots, b_n$, where h and the b_i are atoms, that is, they are expressions of the form $p(t_1, \ldots, t_n)$ where p/n is a *predicate symbol* of arity n and the t_i are *terms*. Terms are constants, variables, or structured terms. Structured terms are expressions of the form $f(t_1, \ldots, t_k)$, where f/k is a functor symbol of arity k and $t_1, \ldots, t_k$ are terms. h is also called the *head* of the clause, and $b_1, \ldots, b_n$ its *body*. Intuitively, a clause represents that the head h will hold whenever the body $b_1, \ldots, b_n$ holds. As an example, consider the definite clause

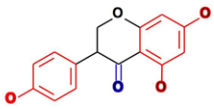
$$mutagenic(M) \leftarrow aromatic\_ring(M, R), atom(M, A, cl),$$

which indicates that a molecule is mutagenic if it contains an aromatic ring structure and a chlorine atom. Clauses with an empty body ($n = 0$) are called *facts*, as in $mutagenic(\texttt{m1})$, indicating that the molecule `m1` is mutagenic.

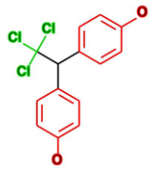
Typically, ILP systems start from a set of positive and negative examples $\mathcal{D}$ in the form of true and false facts, and a background theory $\mathcal{B}$ in the form of a set of definite clauses. The goal is then to induce a hypothesis H (a set of definite clauses) belonging to some hypothesis space $\mathcal{H}$, such that H covers all the positive and none of the negative examples in $\mathcal{D}$. Various notions of coverage can be employed. The most widely used setting is learning from entailment, where an example e is covered if and only if $\mathcal{B} \cup H \models e$, that is, if the example is logically entailed by the hypothesis and the background knowledge.

Example 1 Consider representing the graph structure of molecules—as, for example, in the mutagenicity experiments presented by Srinivasan et al. (1996)—by the following background theory:

```
atm(m1,a1_1,o).      bond(m1,a1_1,a1_2,1).
atm(m1,a2_2,c).      bond(m1,a1_2,a1_3,2).
atm(m1,a2_3,c).      bond(m1,a1_3,a1_4,1).
...                  ...
atm(m1,a2_20,o).     bond(m1,a1_19,a1_20,1).
```



```
atm(m2,a1_1,cl).     bond(m2,a1_1,a1_3,1).
atm(m2,a2_2,cl).     bond(m2,a1_2,a1_3,1).
atm(m2,a2_3,c).      bond(m2,a1_3,a1_4,1).
...                  ...
atm(m2,a2_21,o).     bond(m2,a1_20,a1_21,1).
```



Labels for the examples m1, m2 could be $\mathcal{D} = \{mutagenic(\texttt{m1}), mutagenic(\texttt{m2})\}$. A possible hypothesis $H = \{c_1, c_2, c_3\}$ for this domain is given by the following three clauses:

$$c_1 : \texttt{p}(M) \leftarrow \texttt{aromatic\_ring}(M, [A0, A1, A2, A3, A4, A5]), \texttt{bond}(M, A5, A6, 1), \texttt{atm}(M, A6, \texttt{o}),$$

$$c_2 : \texttt{p}(M) \leftarrow \texttt{atm}(M, A, \texttt{o}), \texttt{bond}(M, A, 2),$$

$$c_3 : \texttt{p}(M) \leftarrow \texttt{atm}(M, A1, \texttt{cl}), \texttt{bond}(M, A1, A0, 1),$$
$$\texttt{atm}(M, A2, \texttt{cl}), \texttt{bond}(M, A2, A0, 1), \texttt{atm}(M, A3, \texttt{cl}), \texttt{bond}(M, A3, A0, 1)$$

where $aromatic\_ring(M, [A0, A1, A2, A3, A4, A5])$ is a background predicate indicating an aromatic ring structure, and we have abbreviated $mutagenic(M)$ with $p(M)$. The hypothesis H covers both examples: example m1 is covered by the clauses c_1 and c_2, and example m2 is covered by the clauses c_1 and c_3 (highlighted in red, blue and green in the figure above).

The outlined ILP learning setting can be summarized as follows:

Problem 1 (Learning from Entailment)

Given

- a background theory $\mathcal{B}$, in the form of a set of definite clauses $h \leftarrow b_1, \ldots, b_n$;
- a set of positive and negative examples $\mathcal{D}$, in the form of positive and negative facts;
- a language of hypotheses $\mathcal{H}$, which specifies the allowed hypotheses;

Find a hypothesis $H^* \in \mathcal{H}$ that covers all positive and no negative examples from $\mathcal{D}$.

Noise in the data can be handled by relaxing the perfect coverage requirement for the hypothesis into the following maximization problem: find

$$H^* = \max_{H \in \mathcal{H}} S(H, \mathcal{D}, \mathcal{B}) \tag{1}$$

where S is an appropriate scoring function evaluating the quality of candidate hypotheses, such as accuracy.

Learning in ILP thus involves a search in a discrete space $\mathcal{H}$ of hypotheses, which is a computationally hard problem in general. However, one can exploit that the hypothesis space is structured by a (partial) generality relation $\preceq$: a hypothesis H_1 is more general than a hypothesis H_2 ($H_1 \preceq H_2$) if and only if any example covered by H_2 is also covered by H_1. We can thus distinguish more general and more specific hypotheses, and this information can be used to prune the search space. For instance, no specializations of a hypothesis h should be considered if we have determined that h is already too specific. More generally, search in the hypothesis space can be conducted in a general-to-specific or specific-to-general fashion, using an appropriate refinement operator (Muggleton and De Raedt 1994; De Raedt 2008) that generalizes or specializes hypotheses. Finally, the computational cost of searching in a discrete space typically forces one to resort to heuristic search algorithms, such as (variations of) greedy search (see De Raedt 2008 for more details). As an example for a greedy general-to-specific ILP learner, we will discuss the FOIL algorithm in Sect. 3.4 (see also Algorithm 1).

The main advantage of ILP techniques is the use of an expressive general purpose representation formalism that enables us (1) to deal with complex structured data, (2) to incorporate prior domain knowledge in the learning process, and (3) to obtain hypotheses which are easily understood by human experts.

2.2 Statistical learning with kernels

In the usual statistical learning framework (see, e.g., Cucker and Smale 2002 for a thorough mathematical introduction) a supervised learning algorithm is given a training set of input-output pairs $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. The set $\mathcal{X}$ is called the input (or instance) space and can be any set. The set $\mathcal{Y}$ is called the output (or target) space; in the case of binary classification $\mathcal{Y} = \{-1, 1\}$ while in the case of regression $\mathcal{Y}$ is the set of real numbers. A fixed (but unknown) probability distribution P on $\mathcal{X} \times \mathcal{Y}$ links input objects to their output target values. The learning algorithm outputs a function $f : \mathcal{X} \to \mathcal{Y}$ that approximates the probabilistic relation between inputs and outputs. A loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ measures the loss incurred in predicting $f(x)$ for an example pair (x, y), and its integral over $X \times Y$ (weighted according to P) measures the expected risk or generalization error of f. Such expected risk cannot be employed for training as P is unknown, and learning algorithms usually minimize a (regularized) empirical risk measured on the training examples. Furthermore, the desired loss can result in a too hard optimization problem, though approximations can sometimes be used instead. For instance, the 0-1 classification loss often leads to NP-hard problems (Höffgen et al. 1995), but it can be upper-bounded by various convex loss functions such as the hinge ($\max(0, 1 - yf(x))$) or the exponential ($\exp(-yf(x))$) loss (Bartlett et al. 2006).

Kernel-based approaches are one of the most popular techniques within the statistical learning framework. A (Mercer) kernel is a positive semi-definite symmetric function[1] $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that generalizes the notion of inner product to arbitrary domains (see, e.g., Shawe-Taylor and Cristianini 2004 for details). When using kernel methods in supervised learning, the space of candidate functions, denoted $\mathcal{F}_K$, is the so-called reproducing kernel Hilbert space (RKHS) associated with K. Learning consists of solving the following

[1] A symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a *positive semi-definite kernel* iff $\forall m \in \mathbb{N}, \forall x_1, \ldots, x_m \in \mathcal{X}$, $\forall a_1, \ldots, a_m \in \mathbb{R}$, $\sum_{i,j=1}^{m} a_i a_j K(x_i, x_j) \geq 0$.

Tikhonov regularized problem:

$$f = \operatorname*{argmin}_{h \in \mathcal{F}_K} C \sum_{i=1}^{m} \ell(y_i, h(x_i)) + \|h\|_K^2 \tag{2}$$

where $\ell(y, h(x))$ is a positive function measuring the loss incurred in predicting $h(x)$ when the target is y, C is a positive regularization constant, $\|\cdot\|_K$ is the norm in the RKHS, and $\Omega : [0, \infty) \to \mathbb{R}$ is a strictly monotonic increasing function acting as regularizer. The representer theorem (Kimeldorf and Wahba 1970) shows that the solutions to the above problem can be expressed as a linear combination of the kernel between individual training examples x_i and x as follows:

$$f(x) = \sum_{i=1}^{m} c_i K(x, x_i). \tag{3}$$

By choosing both the loss function ℓ and the regularizer Ω to be convex, a convex optimization problem is obtained. Popular algorithms in this framework include support vector machines (SVM) (Cortes and Vapnik 1995) and kernel ridge regression (Poggio and Smale 2003; Shawe-Taylor and Cristianini 2004). Equation (3) also encompasses the solution found by other algorithms such as the kernel perceptron (Freund and Schapire 1999).

Advantages of statistical learning with kernels include (1) a principled and robust approach to handling noisy data, (2) effective and efficient techniques to solve the convex optimization problem (Platt 1999; Joachims 1999), and (3) theoretical insight into the generalization performance of certain classes of statistical classifiers (Cortes and Vapnik 1995).

3 Learning logical kernels

3.1 A framework for statistical logical learning

The integration of ILP and statistical learning has the appealing potential of combining the advantages of the respective approaches, namely the expressivity and interpretability of ILP with the effectiveness and efficiency of statistical learning as well as its ability to deal with other tasks than standard binary classification. The integration usually consists of representing examples as input-output pairs as in Sect. 2.2 and replacing the covers definition with a generic function $f(x; H, \mathcal{B}) : \mathcal{X} \to \mathcal{Y}$, mapping input to output values. This implies that the covers function is no longer binary and rather than representing examples as facts $p(x)$ for a predicate p/n that can be true or false, we now use the identifier x to refer to examples. The generic maximization problem can be stated as:

$$\max_{H \in \mathcal{H}} \max_{f \in \mathcal{F}_H} S(f, \mathcal{D}, \mathcal{B}) \tag{4}$$

where $\mathcal{F}_H$ is the set of all functions that can be represented based on the hypothesis H. Standard learning from entailment can be recovered by fixing f to be the coverage function,

$$f(x; H, \mathcal{B}) = \begin{cases} 1 & \text{if } \mathcal{B} \cup H \models p(x), \\ -1 & \text{otherwise} \end{cases}$$

where $p(x)$ is the representation of the example x as a fact, and setting

$$S(f, \mathcal{D}, \mathcal{B}) = -\frac{1}{m} \sum_{i=1}^{m} \mathbb{I}[f(x_i; H, \mathcal{B}) \neq y_i]$$

where $\mathbb{I}$ is a 0-1 value indicator function that computes the 0-1 loss.

Note that Problem (4) consist of jointly optimizing the logical hypothesis and the function $f(x; H, \mathcal{B})$ that replaces logical coverage. In the following, we will refer to the outer optimization problem as *hypothesis learning* and the inner optimization problem as *function learning*. As discussed in Sect. 2.1, hypothesis learning implies searching in a discrete space of candidates, which is a complex task. Thus, heuristic strategies will be employed. In contrast, function learning takes place in a continuous space, for which principled search techniques are available. It is thus unclear whether the scoring function employed for function learning is also suitable for hypothesis learning. In fact, statistical relational learning systems often employ different scoring functions for learning the logical model structure and the statistical part of the model. Problem (4) should therefore be generalized to the following formulation:

$$\max_{H \in \mathcal{H}} S_O\left(\operatorname*{argmax}_{f \in \mathcal{F}_H} S_I(f, \mathcal{D}, \mathcal{B}), \mathcal{D}, \mathcal{B}\right) \tag{5}$$

where S_O and S_I are the scoring functions used for hypothesis and function learning respectively. Generally speaking, this decomposition of the overall optimization problem into a structural and a parameter/function learning problem appears in many statistical relational learning systems. Examples for outer scoring functions S_O proposed in the literature include likelihood (Kersting and De Raedt 2007), pseudo-likelihood (Kok and Domingos 2005), conditional likelihood (Landwehr et al. 2005), AUC-PR (Davis et al. 2005), and AUC-ROC (Frasconi et al. 2008). Examples for proposed inner scoring functions include likelihood (Davis et al. 2005), pseudo-likelihood (Kok and Domingos 2005), and hinge loss (Landwehr et al. 2006). We will discuss in more detail how existing approaches can be cast in the formulation of (5) in Sect. 4.1.

3.2 Logical kernel machines

We now discuss how kernel methods can be naturally used to represent the statistical coverage function $f(x; H, B)$ defined above. The idea is to write f, as in (3), as a linear combination of kernel functions over pairs of training examples, given the hypothesis[2] and background knowledge:

$$f(x; H, \mathcal{B}) = \sum_{i=1}^{m} c_i K(x, x_i; H, \mathcal{B}). \tag{6}$$

The only prerequisite at this point is a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for the relational example space $\mathcal{X}$. The proposed setting allows us to employ ILP techniques to address any learning task amenable to kernel machine algorithms, including binary and multi-class classification, regression, ranking and novelty detection. For instance, using the standard

[2]Note that f is typically named hypothesis in the statistical setting, while we will adhere to ILP terminology and refer to the set of clauses H induced by the algorithm as the hypothesis.

support vector method for classification, the decision function is expressed as

$$\text{sign}(f(x; H, \mathcal{B})) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i K(x, x_i; H, \mathcal{B}) + b\right) \tag{7}$$

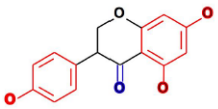
where we generalize (6) to include a bias term b, and take the sign of the function to output a binary decision. Similarly, using support vector regression, the regression function is expressed as

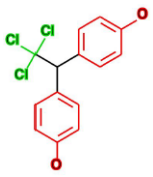
$$f(x; H, \mathcal{B}) = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) K(x, x_i; H, \mathcal{B}) + b. \tag{8}$$

3.3 Kernel functions based on definite clause sets

The simplest way to introduce kernels $K(x_1, x_2, H, \mathcal{B})$ based on a set H of definite clauses is to propositionalize the examples x_1 and x_2 using H and $\mathcal{B}$ and employ existing kernels on the resulting vectors. We will thus map each example x onto a vector $\varphi_{H,\mathcal{B}}(x)$ over $\{0, 1\}^n$ with $n = |H|$, having $\varphi_{H,\mathcal{B}}(x)_i = 1$ if $\mathcal{B} \cup \{c_i\} \models p(x)$, where c_i is the i-th clause $c_i \in H$, and 0 otherwise.

Example 2 Reconsider the background theory given in Example 1, describing the structure of the following two molecules,





and the hypothesis $H = \{c_1, c_2, c_3\}$ for this domain given in Example 1:

c_1 : p(M) ← aromatic_ring(M, [$A0$, $A1$, $A2$, $A3$, $A4$, $A5$]), bond(M, $A5$, $A6$, 1), atm(M, $A6$, o),

c_2 : p(M) ← atm(M, A, o), bond(M, A, , 2),

c_3 : p(M) ← atm(M, $A1$, cl), bond(M, $A1$, $A0$, 1),

atm(M, $A2$, cl), bond(M, $A2$, $A0$, 1), atm(M, $A3$, cl), bond(M, $A3$, $A0$, 1).

Clauses c_1, c_2 succeed on the first example and clauses c_1, c_3 on the second (highlighted above in red, blue and green). Consequently, in the feature space spanned by the clauses c_1, c_2 and c_3, the examples are represented as

$$\varphi_{H,\mathcal{B}}(\texttt{m1}) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \qquad \varphi_{H,\mathcal{B}}(\texttt{m2}) = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

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

$$K_L(\texttt{m1}, \texttt{m2}; H, \mathcal{B}) = \langle \varphi_{H,\mathcal{B}}(\texttt{m1}), \varphi_{H,\mathcal{B}}(\texttt{m2}) \rangle = 1,$$

$$K_L(\mathtt{m1},\mathtt{m1};H,\mathcal{B}) = \langle \varphi_{H,\mathcal{B}}(\mathtt{m1}), \varphi_{H,\mathcal{B}}(\mathtt{m1})\rangle = 2,$$
$$K_L(\mathtt{m2},\mathtt{m2};H,\mathcal{B}) = \langle \varphi_{H,\mathcal{B}}(\mathtt{m2}), \varphi_{H,\mathcal{B}}(\mathtt{m2})\rangle = 2.$$

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

The linear kernel introduced in the above example can be formalized in terms of logical entailment:

$$K_L(x_1,x_2;H,\mathcal{B}) = \#ent_{H,\mathcal{B}}(p(x_1)\wedge p(x_2))$$

where $\#ent_{H,\mathcal{B}}(q) = |\{c \in H | \mathcal{B}\wedge\{c\} \models q\}|$ denotes the number of clauses in H that together with $\mathcal{B}$ logically entail the formula q. 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

$$K_P(x_1,x_2;H,\mathcal{B}) = (\#ent_{H,\mathcal{B}}(p(x_1)\wedge p(x_2)) + 1)^d,$$

which amounts to considering conjunctions of up to d clauses that 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

$$K_G(x_1,x_2;H,\mathcal{B}) = \exp\left(-\frac{\#ent_{H,\mathcal{B}}((p(x_1)\vee p(x_2))\wedge\neg(p(x_1)\wedge p(x_2)))}{2\sigma^2}\right)$$

where the argument of $ent_{H,\mathcal{B}}$ can be interpreted as a kind of symmetric difference between the two examples.

3.4 The kFOIL learning algorithm

We now propose a simple approach for solving the outer optimization problem defined by Problem (5) using ILP techniques. As a hypothesis H defines a kernel function in terms of a set of definite clauses, learning H corresponds to learning a kernel function for relational examples. We thus propose to exploit the expressive power of first-order logic to automatically *learn* an appropriate relational kernel for a given domain. Specifically, Problem (5) requires jointly learning the kernel function and solving the function approximation problem in the resulting RKHS.

The kFOIL algorithm will now be presented as a simple instance of the proposed approach. To learn the hypothesis H, kFOIL employs an adaptation of the well-known FOIL algorithm (Quinlan 1990). FOIL essentially implements a *separate-and-conquer* rule learning algorithm in a relational setting, and is one of the most basic and widely used ILP algorithms.

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, FOIL greedily searches for a clause that scores well. To this aim, it employs a general-to-specific hill-climbing search strategy. Let p/n denote the predicate that is being learned. Then the most general clause, which

Algorithm 1 Generic FOIL algorithm

```
Initialize H := ∅
repeat
    Initialize c := p(X_1, ..., X_n) ←
    repeat
        c := argmax_{c' ∈ ρ(c)} S(H ∪ {c'}, D, B)
    until stopping criterion
    H := H ∪ {c}
    D := update(D, H)
until stopping criterion
output H
```

succeeds on all examples, is "$p(X_1, \ldots, X_n) \leftarrow$". 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 specializes a clause $h \leftarrow b_1, \ldots, b_k$ by adding new literals b_{k+1} to the clause, 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 as well.

Search in standard FOIL is based on the notion of logical coverage: the goal is essentially to cover all positive and no negative examples. In the setting developed here, coverage is replaced by a hybrid statistical-logical model. Consequently, Algorithm 1 needs to be adapted. Specifically, three key modifications need to be made. First, the scoring function $S(H \cup \{c'\}, \mathcal{D}, \mathcal{B})$ is changed: S is replaced by the outer scoring function S_O as defined by (5), that is,

$$S(H \cup \{c'\}, \mathcal{D}, \mathcal{B}) = S_O\Big(\operatorname*{argmax}_{f \in \mathcal{F}_{H \cup \{c'\}}} S_I(f, \mathcal{D}, \mathcal{B}), \mathcal{D}, \mathcal{B}\Big). \tag{9}$$

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 $update(\mathcal{D}, H)$ function in Algorithm 1). In a joint statistical-logical model, this notion of coverage is lost, and the training set is not changed between iterations. Therefore, $update(\mathcal{D}, H)$ returns $\mathcal{D}$. 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 there is no improvement in score between two successive iterations.

Adapting Algorithm 1 in this way yields a "wrapper" approach in which candidate clauses are successively evaluated according to the scoring function S_O. This is similar in spirit to hypothesis search in the nFOIL system (Landwehr et al. 2005). By replacing a generative statistical model with a consistent discriminative one, kFOIL improves over its predecessor nFOIL, as shown in Sect. 6.3, while retaining most of the interpretability inherent in selecting a small set of relevant features. However, a disadvantage of kFOIL as compared to nFOIL is the increased computational complexity of the search procedure, as evaluating a candidate hypothesis now requires one to solve the function optimization problem. Two techniques to improve computational complexity will be discussed later. First, computational complexity can be reduced significantly if the wrapper approach is replaced by an *incremental* learning procedure, which exploits similarities in the optimization problems that need to be solved during search (Sect. 5). Second, we explore directly scoring

the kernel function defined by a candidate hypothesis according to *kernel target alignment*, and postpone solving the function optimization problem until the final hypothesis is found (Sect. 3.5). Section 6 will show empirically that for a combination of these two techniques scoring in kFOIL can be performed in linear time in the number of examples.

3.5 Scoring functions for learning logical kernel machines

The natural inner scoring function S_I for (logical) kernel machines is the negated Tikhonov regularized risk as reported in (2). The choice of the loss function ℓ depends on the learning task; for instance, hinge loss for classification or ϵ-insensitive loss for regression. These scoring functions lead to convex optimization problems, and retaining them in the hybrid statistical-logical setting has the advantage that existing highly optimized software packages can be used.

Joint learning of kernels and function parameters has been addressed in a purely statistical setting by learning combinations of simpler kernel functions, whose coefficients are learned by gradient descent (Chapelle et al. 2002), semidefinite programming (Lanckriet et al. 2004; Ong et al. 2005) or regularization (Micchelli and Pontil 2005). In a hybrid statistical-logical setting like the one we propose here, a constructive approach has to be pursued instead, trading optimality for expressiveness and interpretability of the learned relational rules. Preliminary experiments showed that a straightforward use of the inner scoring function S_I also as outer scoring function S_O produced a highly unstable search and bad overall results. This could be due to the difficulty of directly comparing the minima of different optimization problems as defined by the different hypotheses spaces, but the problem deserves further investigation. Anyhow, the hypothesis search space is discrete, and efficient optimization procedures exploiting convexity cannot be applied in the outer optimization in any case. Therefore, we are free to employ as S_O directly the loss function we are ultimately interested in.

Natural choices for S_O are the 0-1 loss or the area under the ROC curve (AUC) for classification problems, and root mean squared error for regression problems. To account for unbalanced class distributions in classification problems we focus on AUC in this paper (see also Provost et al. 1998 for a comparative analysis of accuracy and AUC in classifier evaluation). Interestingly, there are close connections between the hinge loss function (as in S_I) and AUC (as in S_O). It has been observed that standard SVMs, which optimize hinge loss, achieve very good AUC values, indicating that the two optimization criteria are closely related (Rakotomamonjy 2004). More recently, this has also been supported by a theoretical analysis. Steck (2007) introduced a ranking version of the hinge loss function called the *hinge rank loss*, and showed that minimizing hinge rank loss coincides with maximizing AUC in the limit. Standard hinge loss, which is computationally more convenient to optimize, can be seen as the parametric counterpart of the (discrete) hinge rank loss. Empirically it has been shown that optimizing standard hinge loss instead of rank hinge loss (or AUC directly) often yields near-optimal solutions in terms of AUC performance (Steck 2007).

3.5.1 Direct optimization of the kernel function

The computational bottleneck in the approach proposed so far is the training of a kernel machine for each candidate hypothesis. As an alternative, we propose to score a candidate hypothesis H directly by evaluating the kernel function $K(\cdot, \cdot, H, \mathcal{B})$ it defines, and postpone solving the support vector machine problem until the final hypothesis is found. By encoding similarity between pairs of examples, the kernel function already contains valuable information concerning the potential performance of a learning machine that uses it.

More specifically, in a binary classification setting the relatedness of a certain kernel function to the target can be measured by *Kernel Target Alignment* (KTA) (Cristianini et al. 2001), defined as the normalized Frobenius product between the kernel matrix and the matrix representing pairwise target products:

$$A(K, y) = \frac{\langle K, yy^T \rangle_F}{\sqrt{\langle K, K \rangle_F \langle yy^T, yy^T \rangle_F}} \tag{10}$$

where $y \in \{-1, 1\}^m$ is the target vector for m examples, y^T is the transpose of y, and the Frobenius product is defined as $\langle M, N \rangle_F = \sum_{ij} M_{ij} N_{ij}$. The alignment of a kernel matrix is also related to bounds on the maximum performance of a classifier using this kernel (in terms of generalization error) (Cristianini et al. 2001). In the spirit of (5), we will thus treat the alignment score of a kernel function as an indication of the performance a support vector machine can reach using this kernel, and use it to drive the search for hypotheses. A naive computation of the kernel target alignment as given in (10) has complexity $O(n^2)$ where n is the number of examples. However, in Sect. 5 incremental approaches to computing KTA scores for all candidate clauses encountered during clause search will be discussed, which yield significant computational savings.

3.6 Multi-task statistical logical learning with kernels

Multi-task learning is a technique for solving multiple related tasks in a given domain by learning a joint model for all tasks (Argyriou et al. 2007; Caruana 1997; Evgeniou et al. 2005). The rationale behind multi-task learning is the following. Given limited training data for each individual task, there are typically many (single-task) models that fit the data equally well, and the learner has to rely on its built-in bias to choose between them. By learning a joint model for all tasks, an additional bias is introduced, as the model now has to fit the observed data from all tasks simultaneously. This can significantly alleviate problems associated with sparse training data such as overfitting and unstable search. Empirically, it has been shown that multi-task learning often leads to significantly improved generalization on unseen examples (Caruana 1997).

In our framework for learning logical kernels, we explicitly learn a feature representation, and thus a kernel function, by means of the induced hypothesis H. A natural approach for multi-task learning in the spirit of Caruana (1997) is therefore to share this representation (or, equivalently, the kernel function) across tasks. This can be achieved by an appropriate multi-task scoring function, which is obtained as a combination of single-task scoring functions on the individual tasks. Assume that $S_O(f, \mathcal{D}, \mathcal{B})$ is an (outer) scoring function as introduced in Sects. 3.1 and 3.5, and that $\mathcal{D}_1, \ldots, \mathcal{D}_T$ are the available training data for M tasks $T_1, \ldots, T_M$. A simple but effective multi-task scoring function is obtained by averaging single-task scores, that is, by replacing (5) with

$$\max_{H \in \mathcal{H}} \frac{1}{T} \sum_{t=1}^{T} S_O\Big(\operatorname*{argmax}_{f \in \mathcal{F}_H} S_I(f, \mathcal{D}_t, \mathcal{B}), \mathcal{D}_t, \mathcal{B}\Big). \tag{11}$$

Experimental results presented in Sect. 6 show three advantages of multi-task learning in the proposed setting. First, they show a consistently improved generalization performance, confirming the advantages of multi-task learning observed in the literature (Caruana 1997). Second, in our setting learning a shared clause set for multiple tasks leads to a more compact representation of the learned concept, as the multi-task clause set is significantly smaller than

the union of the task-specific clause sets. In terms of the similarity/kernel function learned, this yields a generic definition of similarity that is shared between tasks and should be easier to interpret than a number of task-specific similarity functions. Third, multi-task learning also results in significant computational savings.

Finally, traditional approaches to multi-class learning with support vector machines can be considered from a multi-task perspective. Traditionally, multi-class classification in SVMs is performed by solving several binary classification tasks, either in a one-vs-one or one-vs-rest setup, using a shared representation for the examples which is fixed in advance. Test examples are then classified into the class with the highest margin. In such a setting, we propose to pursue a multi-task approach, by jointly learning the shared representation of examples which best fits all binary tasks. In this case, the average of single-task scoring functions (11) can also be replaced by multi-class accuracy. Multi-class accuracy can thus be seen as an additional multi-task scoring function in this setting. Section 6 shows that also for multi-class classification, multi-task approaches can improve over standard single-task (i.e., one-vs-all) approaches.

4 Related work

4.1 Structure learning in statistical relational learning

Many approaches for structure learning in the statistical relational learning setting can be cast into the framework presented in Sect. 3.1. The framework defines a decomposition of the overall optimization problem into a structural and a parameter or function learning part, with respective scoring functions S_O and S_I. Let us briefly discuss other instantiations of this setting proposed in the literature. Stochastic Logic Programs are learned in Muggleton (2000) by maximizing the Bayesian posterior probability of the program given the examples (S_O), assuming uniform probability distribution for each predicate (S_I), but refining predicate probabilities for the final program according to smoothed counts of their occurrence in example proofs. Bayesian Logic Programs (BLPs) are learned by a probabilistic extension of learning from interpretations (Kersting and De Raedt 2007). Search in the hypothesis space is conducted with a greedy hill-climbing strategy, driven by maximum likelihood for both S_O and S_I. Structure learning in Markov Logic Networks (Kok and Domingos 2005) is conducted either in a best-first or shortest-first fashion, where the latter implies adding all "good" clauses of length ℓ before trying any of length $\ell + 1$. Both S_O and S_I rely on a weighted pseudo-log-likelihood, in which the probability of each grounding of each first-order atom, given its Markov Blanket, is downweighted by the number of groundings. An additional regularization term penalizes large deviations from the initial structure. The nFOIL system uses maximum likelihood for S_I and maximum conditional likelihood for S_O (Landwehr et al. 2005). SAYU uses maximum likelihood for S_I and area under the precision-recall curve for S_0 (Davis et al. 2005). Structural Logistic Regression employs a general-to-specific search strategy driven by a regularized maximum likelihood measure for both scores (Popescul et al. 2003). In the RUMBLE margin-based rule learner, the inner score S_I is named margin minus variance, and is computed as the difference between the empirical mean and variance of the classification margin over training examples, while the outer score S_O is a generalization bound combining the margin minus variance with a capacity term (Rückert and Kramer 2007). Type Extension Trees (TET) are learned using a kind of pseudo-maximum-likelihood for S_I, and a generic score such as the area under the ROC curve for S_O (Frasconi et al. 2008). Finally, in a regression setting, the First-Order Regression System relies on mean squared error for both (negated) outer and inner scores (Karalič and Bratko 1997).

4.2 Logical kernel machines

A number of recent works have addressed the problem of developing relational kernels with logic. Cumby and Roth (2003) described a family of relational kernels defined using a simple description logic. Gärtner et al. (2004) defined kernels over complex individuals using higher order logic. Passerini et al. (2006) introduced the notion of *visiting* predicates that are used to explore relational objects given background knowledge, and they defined kernel between objects as similarities between proof trees obtained for the visitor predicates. Declarative kernels rely on topological and parthood relations to define object similarity in terms of parts and connections between parts (Frasconi et al. 2005). Recently, Wachman and Khardon (2007) proposed a kernel on relational data by upgrading walk-based graph kernels to hypergraphs. Complementary research has focused on defining distances in a relational setting (see Ramon and Bruynooghe 1998; Kirsten et al. 2001 or Ramon 2002 for an extensive treatment). While retaining much of the expressivity granted by logic, these approaches need to pre-define an appropriate kernel function or distance measure for the domain at hand. In contrast, the kernel we introduced here defines similarity in terms of a small set of clauses that are learned from data by leveraging ILP-style search techniques.

4.3 Static and dynamic propositionalization

In *static propositionalization*, a set of features is first computed from the relational data and a propositional learner is then trained on the resulting feature space. The feature set can be computed by an ILP system (as in the SVILP system of Muggleton et al. 2005) or a pattern miner (as by Kramer and De Raedt 2001), or it can be pre-defined without any additional feature construction/selection phase (as in the LINUS system of Lavrac and Dzeroski 1994). The main drawback of static propositionalization approaches is that the feature construction step is decoupled from the actual statistical modeling, and uses a different selection criterion. Thus, the selected feature set will not be optimal for the particular statistical model in which the features are used. Static propositionalization typically also results in large feature sets, reducing the interpretability of the final model. In contrast, we propose a *dynamic propositionalization* approach, in which the feature set defining the kernel and the statistical classifier are optimized jointly. The approach is related to other dynamic propositionalization systems such as SAYU (Davis et al. 2005), nFOIL (Landwehr et al. 2005), and Structural Logistic Regression (Popescul et al. 2003), but relies on kernel methods instead of probabilistic models. This enables us to tackle different learning tasks such as classification or regression in a uniform framework. Also, the resulting kFOIL system has been shown to improve upon nFOIL in terms of predictive accuracy in our experimental study. Trading interpretability and efficiency for effectiveness, the RUMBLE (Rückert and Kramer 2007) margin-based rule learner uses a large set of features encoded by mathematical expressions, frequent substructures, definite clauses and their combinations, adding them one at a time, and keeps the subset achieving the best results, removing features with low weight. On the other hand, kFOIL aims at keeping a simple and parsimonious approach, that can be easily applied to other ILP systems.

Finally, dynamic approaches to propositionalization have also recently been considered in the graph mining community, where the goal is to learn a set of subgraphs that propositionalize graph data. In this context, Saigo et al. (2009) consider a boosting approach to collect a set of informative patterns, which can be seen as a dynamic propositionalization approach combining graph patterns and kernel machines.

4.4 Kernel learning

The technique we propose can also be seen as a kernel learning approach, in that it constructs the kernel based on the available data. Joint learning of kernels and function parameters has been addressed in a purely statistical setting by learning combinations of simpler kernel functions, whose coefficients are learned by gradient descent (Chapelle et al. 2002), semi-definite programming (Lanckriet et al. 2004; Ong et al. 2005) or regularization (Micchelli and Pontil 2005). Ong et al. (2005) propose a general framework where kernels are chosen from a hyper-RKHS induced by hyperkernels based on an appropriately regularized scoring function. These approaches are typically more principled than our approach (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, kFOIL is the first system that can learn kernels defined by small sets of interpretable first-order rules.

4.5 Multi-task learning

Multi-task learning is traditionally addressed by learning a common representation of an example for different tasks. For instance, in feedforward neural networks this can be achieved by sharing the hidden layers across tasks. Multi-task learning with kernel machines was first introduced by Evgeniou et al. (2005), by including in the regularization term a matrix which encodes the relation between tasks. Evgeniou et al. (2005) showed how to convert the resulting problem into an equivalent single-task problem via an appropriate multi-task kernel function. However, this method requires to explicitly encode task relationships, which have to be known in advance. Multi-task kernel learning was later introduced using a generalization of single-task 1-norm regularization (Obozinski et al. 2006; Argyriou et al. 2007), which minimizes the number of non-zero features across tasks, or relying on maximum entropy discrimination (Jebara 2004).

The setting considered in this paper is different, as we are learning a hybrid statistical-logical model. We propose to exploit multi-task information for learning the relational model structure, by constructing a relational kernel function that is shared across tasks. As shown in Sect. 6, this leads not only to better generalization but also yields a more compact representation of the learned concept. To the best of the authors' knowledge, multi-task kernel learning has never been addressed in a statistical relational learning context before. Indeed, multi-task structure learning itself has received little attention in the statistical relational learning setting, a notable exception being a recent work by Deshpande et al. (2007) on learning multi-task probabilistic relational rule sets.

Multi-task learning has also received attention in the ILP community, in the form of learning several related concepts simultaneously. Different approaches have been pursued. Related to our approach is the work by Reid (2004), where the assessment of candidate clauses on the primary task is augmented with the performance of similar rules on a secondary task. Furthermore, a scenario resembling multi-task learning has been studied in Datta and Kibler (1993), where (sub)structures of concepts already learned are used as building blocks when learning a new concept. A further related scenario is that of *repeat learning* and *multiple predicate learning* (Khan et al. 1998; De Raedt et al. 1993), where an ILP learner has to discover a series of related concepts drawn from some (initially unknown) distribution. Moreover, *predictive clustering trees* have been used in an ILP setting. These trees can be used in a multi-task setting, where predictions for several tasks are made at every leaf (Blockeel et al. 1998, 2004).

5 Computational complexity and incremental optimization

The computational bottleneck in kFOIL learning is the successive evaluation of candidate clauses. Assume the current theory is H with $|H| = n$. Candidate clauses are obtained as refinements of the currently best clause c (see Algorithm 1). Thus, assume we are evaluating a candidate clause $c' \in \rho(c)$, and let $H' = H \cup \{c'\}$ denote the hypothesis including c'. Evaluation consists of the following three steps:

1. $\forall x \in \mathcal{D}$, compute the feature space representation $\varphi_{H',\mathcal{B}}(x)$ of x.
2. Compute the kernel matrix M, i.e., $\forall x, x' \in \mathcal{D}$ compute $K(x, x', H', \mathcal{B})$.
3. Compute $S(H', \mathcal{D}, \mathcal{B})$. Here, two cases have to be distinguished.
   (3a) S is a scoring function that requires training a support vector machine, such as 0-1 loss or AUC. In this case, a (soft margin) SVM optimization needs to be performed. In binary classification, for instance, this amounts at maximizing:

$$L(\alpha) = \sum_{i=1}^{k} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j, H', \mathcal{B})$$

$$\text{subject to} \quad C \geq \alpha_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{k} \alpha_i y_i = 0 \tag{12}$$

   where $x_1, \ldots, x_m$ are the examples in $\mathcal{D}$ and $y_1, \ldots, y_m$ the corresponding labels.
   (3b) S is defined by kernel target alignment (cf. Sect. 3.5). In this case, we have to compute the alignment $A(K, y)$ between the kernel $K = K(\cdot, \cdot, H', \mathcal{B})$ and the examples as defined by (10).

In the following, we show how significant computational savings can be obtained in all of the outlined steps by incrementally updating the relevant pieces of information. The pseudocode and detailed description of the corresponding algorithms can be found in the Appendix.

5.1 Incrementally computing the feature space representation

In order to compute the feature space representation $\varphi_{H',\mathcal{B}}(x)$ for every example $x \in \mathcal{D}$, kFOIL has to first retrieve the set of examples $cov(c')$ covered by the clause c', a task that typically has to be carried out in any ILP system. This task can be solved more efficiently in an incremental way. First, we compute $cov(c)$ for the current best clause c before evaluating coverage of all refinements $c' \in \rho(c)$. As c' is a refinement of c, $cov(c') \subseteq cov(c)$. Thus, coverage of c' only has to be checked on $x \in cov(c)$. Second, coverage calculations in kFOIL are sped up additionally by remembering for every free variable V in c and every example $x \in cov(c)$ the set of constant bindings

$$const(V, x) = \{a \in \mathcal{C} \mid c\theta \text{ covers } x, \theta = \{V/a\}\}$$

making c cover the example x, where $\mathcal{C}$ is the set of all (type-conform) constants that can be bound to V. Assume that c' is obtained from c by adding literal l, that is, $c' = c, l$, and $V_1, \ldots, V_r$ are the shared variables between l and c. Now, c' can only cover x if there is a tuple $(a_1, \ldots, a_r) \in const(V_1, x) \times \cdots \times const(V_r, x)$ such that $l\theta$ covers x for substitution $\theta = \{V_1/a_1, \ldots, V_r/a_r\}$. As shown empirically in Sect. 6, this yields significant computational savings. Similar strategies have been used in other ILP systems, for instance, the

original FOIL algorithm also keeps track of tuple assignments satisfying clauses (Quinlan 1990).

A further major speedup can be obtained from the way the kernel function in our approach is defined in terms of a set of clauses. Note that the complex relational example set $\mathcal{D}$ is mapped to the much simpler (feature) space $\varphi_{H',\mathcal{B}}(\mathcal{D}) \subseteq \{0,1\}^n$ in which the kernel function is computed. Two examples $x, x' \in \mathcal{D}$ of the same class are indistinguishable in the feature space representation if $\varphi_{H,\mathcal{B}}(x) = \varphi_{H,\mathcal{B}}(x')$, i.e., they exhibit the same structural features. Thus, a hypothesis H partitions $\mathcal{D}$ into clusters of examples that share the same feature space representation. For the subsequently employed kernel method, the examples in one cluster can be merged to one example with a weight corresponding to the cluster size.

We will refer to the number $\bar{m}$ of clusters as the *effective* number of examples, because this number determines the complexity of the kernel method (including the size of the kernel matrix). A detailed empirical analysis of the computational savings achievable in this way is presented in Sect. 6.8.

5.2 Incrementally computing the kernel matrix

A lower-triangular representation of the kernel matrix M is kept in memory, and incrementally updated as clauses are added to the current hypothesis H. Note that the size of the kernel matrix is quadratic only in the number of *effective* examples $\bar{m}$.

Initially, $H = \emptyset$, $\bar{m} = 1$ and $K(x, x', H, \mathcal{B}) = 0$ for all $x, x' \in \mathcal{D}$. For scoring a candidate clause c', its contribution to the kernel function is computed and a number of cells in M needs to be incremented. If the clause does not cause any cluster in $\mathcal{D}$ to be split, that is, it does not change $\bar{m}$, this can be done easily in time $O(|\varphi(cov(c))|^2)$, where $|\varphi(cov(c))|$ is the effective number of examples covered by c. If c splits a cluster in $\mathcal{D}$, the dimensionality $\bar{m}$ of the kernel matrix increases, and some of the cells need to be split. Splitting an effective example i implies: adding a row of length $\bar{m}+1$ to the lower triangular representation of the kernel matrix; copying $\bar{m}+1$ kernel values into it, setting $M_{\bar{m}j} \leftarrow M_{ij}$ for $j \in [0, \bar{m}-1]$ and $M_{\bar{m}\bar{m}} \leftarrow M_{ii}$; updating the number of effective examples $\bar{m} \leftarrow \bar{m}+1$; incrementing the row entries corresponding to covered effective examples by one. A single split can thus be done in $O(\bar{m})$ time, and a full update in time $O(|\varphi(cov(c))|\bar{m})$, where $\bar{m}$ is the number of effective examples after all splits. Note that this number will typically be much smaller than the overall number of examples, and Sect. 6.8 empirically shows that it indeed grows with its square root in a real world data set.

5.3 Incremental SVM optimization

If S is a scoring function that requires training a support vector machine during clause evaluation, efficiently solving the optimization problem given by (12) is crucial. Well-known algorithms for this problem include the sequential minimal optimization algorithm by Platt (1999) or other chunking techniques (Joachims 1999), and more recently introduced online optimization approaches such as LaSVM (Bordes et al. 2005) or stochastic gradient descent (Shalev-Shwartz et al. 2007). Optimization in the kFOIL implementation is carried out using the SVMlight package, which is based on the technique described by Joachims (1999) (see Sect. 6). Assume we have already solved the optimization problem for the currently optimal clause c, and are evaluating a refinement $c' \in \rho(c)$. That is, we have maximized (12) for the feature space representation resulting from the hypothesis $H \cup \{c\}$. Let α^0 denote an optimal solution. Typically, the optimization problem resulting from the hypothesis $H \cup \{c'\}$ will be very similar: in the feature space representation of the data, only one of the attributes has

been changed, and only on a subset of the examples (as only one feature has been refined). One can thus expect that also the optimal solution α^* for the new problem will be close to the old solution α^0. A straightforward but effective approach to incremental optimization is thus to restart the optimization at the old maximum α^0, and continue optimizing until the new optimality criterion (Karush-Kuhn-Tucker conditions) is met. Intuitively, this corresponds to starting from the old separating hyperplane and slightly adapting it until it defines the max-margin solution of the new optimization problem. Section 6 will show significant benefits of this technique compared to re-starting the optimization procedure from scratch.

5.4 Incrementally computing kernel target alignment

Finally, also the kernel target alignment score as defined in (10) can be computed incrementally. The key observation is that as a clause c is added to H, it produces an additional, incremental contribution to the kernel matrix M. This contribution can be propagated to the three Frobenius norms from which KTA is computed (see (10)). For all examples covered by the candidate clause c, their contribution to the previously computed norms should be first removed, and then replaced with the contribution due to their updated kernel values.

6 Experimental evaluation

This section presents an experimental evaluation of the proposed kFOIL algorithm in several domains. The goal of the experimental study is two-fold. In a first part, the algorithm is compared to several related systems that employ ILP and propositionalization approaches, in a similar setup as reported in Landwehr et al. (2006). In a second part, we present a detailed analysis of the performance of the algorithm in different learning settings and with different scoring functions. Specifically, we compare multi-task and single-task learning, evaluate different scoring functions with regard to accuracy of inferred models and computational cost, and explore different approaches to multi-class classification. Finally, the computational complexity and scaling behavior of kFOIL will be investigated, and the clause sets returned by the algorithm will be inspected.

6.1 Experimental domains and data sets

Table 1 gives an overview of the different data sets used in the experimental evaluation. Most of the domains are concerned with *structure-activity relationship* (or *SAR*) problems. SAR problems are of central importance in many areas of bio- and chemoinformatics: given information about the chemical structure of a substance, the task is to predict its activity with regard to a certain property of interest. This property can be to activate or block a receptor in the human body, toxicity, suppression of tumor growth or more generally activity as a pharmacological agent. The search for substances with such properties currently relies on large-scale screening trials (so-called *bioassays*), which measure the activity of thousands of chemical compounds. They generate large quantities of experimental data, which are stored in centralized, easily accessible public databases such as PubChem[3] (Wheeler et al. 2008). Such data provides interesting opportunities for machine learning, particularly for techniques that can learn from structured data: if good models that relate compound

[3]The PubChem database is available at http://pubchem.ncbi.nlm.nih.gov/sources/.

Table 1 Overview of all data sets used in experiments, including the number of classes, number of available examples, accuracy of majority class predictor, number of relations that are used in rules, and the number of facts in the Prolog database

Data Set	#Classes	#Examples	Maj. Class	#Relations	#Facts
Mutagenesis r.f.	2	188	66.5%	4	10324
Mutagenesis r.u.	2	42	69.1%	4	2109
Alzheimer amine	2	686	50.0%	20	3754
Alzheimer toxic	2	886	50.0%	20	3754
Alzheimer acetyl	2	1326	50.0%	20	3754
Alzheimer memory	2	642	50.0%	20	3754
NCTRER	2	232	56.5%	3	9283
BioDeg—classification	2	328	56.4%	36	27236
BioDeg—regression	n.a.	328	n.a.	36	27236
NCGC BJ (AID 421)	2	1285	96.2%	29	89929
NCGC Jurkat (AID 426)	2	1242	89.1%	29	89929
NCGC Hek293 (AID 427)	2	1250	94.1%	29	89929
NCGC HepG2 (AID 433)	2	1282	96.1%	29	89929
NCGC MRC5 (AID 434)	2	1289	96.2%	29	89929
NCGC SK-N-SH (AID 435)	2	1281	93.3%	29	89929
MTDP E.coli (AID 365)	2	206	51.2%	23	23734
MTDP Human (AID 366)	2	206	79.5%	23	23734
MTDP HIV-2 (AID 367)	2	206	73.7%	23	23734
NCI BT_549	2	2778	50.4%	30	283612
NCI HCC_2998	2	3177	56.8%	30	283612
NCI HS_578T	2	2870	54.0%	30	283612
NCI SR	2	3006	62.2%	30	283612
NCI T_47D	2	2909	53.3%	30	283612
WebKB	6	1089	51.2%	6	86392

structure to activity can be built, some of the tests currently performed in laboratories could be replaced by automatic classification, greatly accelerating the screening process. Bioassay data is also a natural application area for multi-task learning, as substances have often been tested for several related properties, and thus come with several class labels. Jointly building a model for all properties can yield increased predictive accuracy, as will be shown below.

The individual data sets will now be described in more detail, and their use in the study will be motivated. The *Mutagenesis* data sets are concerned with predicting the mutagenicity of small molecules based on their chemical structure, and are one of the best known ILP benchmark data sets (Srinivasan et al. 1996). We use the version in which only atom and bond structure is available. For *Alzheimer* (King et al. 1995), the aim is to compare analogues of Tacrine, a drug against Alzheimer's disease, according to four desirable properties: inhibit *amine* re-uptake, low *toxicity*, high *acetyl* cholinesterase inhibition, and good *reversal* of scopolamine-induced memory deficiency. Examples consist of pairs (c_1, c_2) of two analogues, and are labeled positive if and only if c_1 is rated higher than c_2 with regard to the

property of interest. The relation is transitive and anti-symmetric but not complete (for some pairs of compounds the result of the comparison could not be determined). The *NCTRER* data set has been extracted from the EPA's DSSTox NCTRER Database (Fang et al. 2001). It contains structural information (atoms and bonds) for a diverse set of natural, synthetic and environmental estrogens, and classifications with regard to their binding activity for the estrogen receptor. In the *Biodegradability* domain the task is to predict the biodegradability of chemical compounds based on their molecular structure and global molecular measurements (Blockeel et al. 2004). The original (numeric) target variable is the half-life for aerobic aquenous biodegradation of the particular chemical compound. Alternatively, the problem can be treated as a classification task by thresholding this target variable. Mutagenesis, Alzheimer, NCTRER and Biodegradability were included in the study because they are well-known benchmark data sets for ILP and relational learning methods, and to build upon the experiments reported in Landwehr et al. (2006). Furthermore, Alzheimer can be cast as a multi-task domain, with tasks corresponding to the four properties of interest.

The following additional SAR problem domains were chosen because they are natural test-cases for multi-task learning. The *NCGC* data sets contain results of high throughput screening assays to determine in vitro cytotoxicity of small molecules. Experiments have been performed multiple times with cell lines derived from different tissue types: BJ cell line (human foreskin fibroblasts), Jurkat cell line (human T cell leukemia), Hek293 cell line (human embryonic kidney cells), HepG2 cell line (hepatocellular carcinoma), MRC5 cell line (human lung fibroblasts) and SK-N-SH cell line (human neuroblastoma). Test results for different cell lines will typically be different but related, thus it is natural to treat them as different prediction tasks in a multi-task learning setting. The *MTDP* data sets contain results of enzymatic assays for inhibition of *ribonuclease H* activity. Individual data sets represent assay results for ribonuclease H enzymes from different organisms: E. coli ribonuclease H, human ribonuclease H1, and HIV-2 ribonuclease H. Again, they can be treated as different but related prediction tasks. Both the NCGC and MTDP data sets have been extracted from the PubChem database. Compound descriptions and class labels, as well as more details about the experimental protocol, are available from this database by looking up the bioassay ID (denoted *AID XXX* in Table 1). Finally, the *NCI* data sets provide screening results for the ability of compounds to suppress or inhibit the growth of tumor cells (Swamidass et al. 2005). Screening results are available for 60 different cell lines; however, not all compounds have been tested against all cell lines. We selected five cell lines (BT_549, HCC_2998, HS_578T, SR, T_47D) that result in relatively small data sets but together contain test results for most compounds used in the study. The NCI data sets are also significantly larger than the other domains considered, and thus serve as a benchmark to evaluate the computational efficiency of kFOIL. In the NCGC, MTDP and NCI domain, background knowledge is supplied encoding potentially relevant molecular substructures, such as rings and common functional groups (carboxyl, hydroxyl, nitro, etc.).

WebKB is a multi-class domain which has a different background and will be discussed in more detail in Sect. 6.6.

6.2 kFOIL implementation and experimental setup

The kFOIL algorithm, with computational improvements as discussed in Sect. 5 and support for multi-task learning as described in Sect. 3.6 has been implemented based on YAP Prolog[4] and the support vector machine package SVMlight.[5] The implemented system will be

[4]For more information, see http://www.dcc.fc.up.pt/~vsc/Yap/.

[5]For more information, see http://svmlight.joachims.org/.

denoted as KFOIL,[6] and in general we use small caps font to denote implemented systems (in contrast to their underlying algorithms).

Experiments with a prototype implementation of kFOIL were already presented in Landwehr et al. (2006), in a single-task setting and using the first 9 of the 24 data sets listed in Table 1. Besides using additional (and larger) data sets, the present study also considers multi-task learning, different scoring functions, multi-class classification, and an analysis of the computational complexity of our approach. Moreover, in the experiments presented in Landwehr et al. (2006), several algorithm parameters were pre-specified: a threshold for detecting convergence, a limit on the number of clauses allowed in a model, and a fixed regularization parameter C for the SVM training. It is not obvious how these parameters interact, especially in controlling overfitting, and how they could be jointly optimized. Thus, we have made an effort to reduce the number of parameters that need to be pre-specified in the new experiments. There is no limit on the number of clauses that can be used in a model, and no convergence threshold. Instead, the model is refined as long as the training score improves (even by a small margin). To avoid overfitting, we perform model selection to choose the regularization parameter C by (repeatedly) splitting the available training data into a training and a validation set. As in the earlier experimental study, a polynomial kernel of degree $d = 2$ is used in all experiments, and a beam size of 5 is used unless noted otherwise.

Evaluation of algorithms is performed by cross-validation or more generally multiple splits into train/test data. As kFOIL can produce a ranking (via example margins in the SVM model), area under the ROC curve (AUC) can be computed on the test data. For binary classification problems, performance is mostly evaluated by AUC, except when directly comparing against accuracy results from the literature. The motivation for using AUC instead of accuracy is that some of the data sets used in the study are very unbalanced, and there are in general arguments for preferring AUC over accuracy as an evaluation measure (Provost et al. 1998). For regression problems, performance is evaluated by root mean squared error (RMSE).

6.3 Comparison to ILP and propositionalization approaches

We begin by comparing KFOIL to other relational learning approaches, replicating the experiments presented in Landwehr et al. (2006). More specifically, we considered the following systems. NFOIL (Landwehr et al. 2005) is a dynamic propositionalization system that combines naive Bayes and FOIL in a similar spirit as KFOIL combines kernels and FOIL. ALEPH is a state-of-the-art ILP system developed by Ashvin Srinivasan.[7] It is based on the concept of *bottom clauses*, which are maximally specific clauses covering a certain example. Furthermore, we consider a static propositionalization baseline based on a variant of the relational frequent query miner WARMR (Dehaspe et al. 1998), as WARMR patterns have shown to be effective propositionalization techniques on similar benchmarks (Srinivasan et al. 1999). The variant used was C-ARMR (De Raedt and Ramon 2004), which focuses on so-called free patterns (more details about the experimental setup can be found in Landwehr et al. 2006). Table 2 presents cross-validated accuracy results for KFOIL, NFOIL, ALEPH and C-ARMR. To facilitate comparison to Landwehr et al. (2006), KFOIL is run with accuracy scoring. Results clearly show that KFOIL outperforms ALEPH and

[6]The implementation is available from http://www.cs.uni-potsdam.de/~landwehr/kfoil/kfoil.tgz.

[7]More information on Aleph can be found at http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html.

Table 2 Average predictive accuracy on Mutagenesis, Alzheimer and NCTRER for KFOIL with accuracy scoring, 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 data set, explains the high variance of the results), on all other data sets a 10 fold cross-validation. • indicates that the result for kFOIL is significantly better than for other method (paired two-sided t-test, $p = 0.05$)

Data Set	KFOIL	NFOIL	ALEPH	C-ARMR
Muta. r.f.	77.0 ± 14.5	75.4 ± 12.3	73.4 ± 11.8	73.9 ± 11.2
Muta. r.u.	85.7 ± 35.4	78.6 ± 41.5	85.7 ± 35.4	76.2 ± 43.1
Alz. amine	89.8 ± 5.7	86.3 ± 4.3	70.2 ± 7.3•	81.2 ± 4.5•
Alz. toxic	90.0 ± 3.85	89.2 ± 3.4	90.9 ± 3.5	71.6 ± 1.9•
Alz. acetyl	90.6 ± 3.4	81.2 ± 5.2•	73.5 ± 4.3•	72.4 ± 3.6•
Alz. mem.	80.5 ± 6.2	72.9 ± 4.3•	69.3 ± 3.9•	68.7 ± 3.0•
NCTRER	78.5 ± 9.3	78.0 ± 9.1	50.9 ± 5.9•	65.1 ± 13.2•

Table 3 Result on the Biodegradability data set. 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 by Blockeel et al. (2004). For classification, average accuracy is reported, for regression, root mean squared error. • indicates that the result for KFOIL is significantly better than for other method (unpaired two-sided t-test, $p = 0.05$)

Data Set	KFOIL	TILDE	S-CART
Classification			
BioDeg GR	73.4 ± 1.63	73.6 ± 1.1	72.6 ± 1.1
BioDeg GP1P2R	73.5 ± 0.95	72.9 ± 1.1	71.3 ± 2.3
Regression			
BioDeg GR	1.139 ± 0.036	1.265 ± 0.033•	1.290 ± 0.038•
BioDeg GP1P2R	1.182 ± 0.038	1.335 ± 0.036•	1.301 ± 0.049•

static propositionalization. Moreover, it consistently improves upon its predecessor NFOIL, indicating that in a hybrid statistical-relational setting kernel methods can improve over a simple naive Bayes model.

Table 3 compares KFOIL to the relational tree-induction algorithms TILDE and S-CART on the Biodegradability classification and regression data sets. KFOIL has been run with accuracy (classification problem) and RMSE (regression problem) scoring. As in the experiments reported in Blockeel et al. (2004), results are averaged over five runs of a ten-fold cross-validation, and the same splits into training and test sets have been used. KFOIL clearly outperforms the other two approaches in the regression setting, while yielding comparable accuracy in the classification setting.

6.4 Evaluation of different scoring functions

As discussed in Sect. 3.2, different scoring functions can be considered for classification problems. From a computational perspective, the main difference is between 0-1 loss or AUC on the one hand and kernel target alignment on the other hand. The former scores require to solve the SVM optimization problem (12), while kernel target alignment can be

Table 4 Average cross-validated prediction accuracy of kFOIL with accuracy scoring and standard beam size 5 (Acc[5]) and kFOIL with KTA scoring and beam sizes 5, 10, 20 and 30 (KTA[5], KTA[10], KTA[20], KTA[30])

Data Set	Acc[5]	KTA[5]	KTA[10]	KTA[20]	KTA[30]
Mutag. r.f.	77.0 ± 13.7	74.9 ± 12.1	74.4 ± 12.0	77.7 ± 11.7	78.8 ± 6.6
Mutag. r.u.	85.7 ± 35.0	83.3 ± 37.3	83.3 ± 37.3	85.7 ± 35.0	85.7 ± 35.0
Alzh. amine	89.8 ± 5.4	72.7 ± 6.3	75.5 ± 5.8	82.5 ± 6.1	83.1 ± 5.2
Alzh. toxic	90.0 ± 3.7	80.9 ± 4.3	88.9 ± 4.1	92.3 ± 1.6	93.7 ± 1.1
Alzh. acetyl	90.6 ± 3.2	74.7 ± 3.6	75.0 ± 3.0	81.7 ± 4.3	83.2 ± 3.5
Alzh. memory	80.5 ± 5.9	66.8 ± 8.0	63.9 ± 7.6	75.1 ± 4.2	77.4 ± 3.9
NCTRER	78.5 ± 8.8	78.5 ± 8.8	78.5 ± 8.8	78.0 ± 9.3	77.6 ± 8.9
BioDeg GR	73.4 ± 1.6	63.5 ± 1.7	64.8 ± 0.8	65.1 ± 1.9	69.1 ± 1.6
BioDeg GP1P2R	73.5 ± 0.9	68.1 ± 2.1	68.5 ± 1.1	68.6 ± 2.1	68.6 ± 2.0

directly computed given the kernel matrix (10), which is typically much more efficient (see also Sect. 6.8).

Table 4 compares KTA scoring to accuracy scoring on the data sets used in Landwehr et al. (2006). Results show that if the standard beam size of 5 is used, KTA is substantially less accurate. There are two possible explanations for this result: either kernel target alignment is generally not a good scoring function in our context, or it does not work well together with the greedy search strategy employed in kFOIL. In the latter case, the clause set truly maximizing KTA would result in a good joint model, but only a (strongly) suboptimal clause set is found during search. To test this hypothesis, the system was run with increased beam sizes (10, 20, and 30). This improves relative performance, indicating that the weak results for beam size 5 are at least partly due to local search issues. For the rest of the experimental study, we thus use a beam size of 20 rather than the standard 5 for kFOIL with KTA scoring. Even with the increased beam size KTA scoring is typically more efficient than accuracy or AUC scoring (see Sect. 6.8).

6.5 Single-task vs. multi-task learning

For the four multi-task domains (Alzheimer, NCGC, MTDP and NCI), the benefits of simultaneously learning a joint model for all tasks were explored. In these domains, there are between three and six different target labels available, although not all labels are necessarily known for all examples (cf. Table 1). We compare a multi-task (MT) approach against a single-task (ST) approach using the following methodology. All examples available in a given domain are first split into a training set (80%) and test set (20%). In the MT approach, a joint model is built from the training set using the technique described in Sect. 3.6. In the ST approach, one model is built for each task using those examples for which label information for that task is available. For every example in the test set, both approaches return a predicted class label for every task. This prediction is compared to the true label for that task if it is known, and resulting area under the ROC curve (AUC) is computed. To obtain reliable estimates, results are averaged over 50 random splits into training and test set.

Table 5 shows average AUC results on Alzheimer, NCGC, MTDP and NCI for kFOIL in the single-task and multi-task setting, using KTA and AUC scoring. For the NCI data

Table 5 Average AUC ± standard deviation on Alzheimer, NCGC, MTDP and NCI for KFOIL with KTA and AUC scoring, using a single-task (ST) or multi-task (MT) learning setting. Results are averaged over 50 random 80%/20% train/test splits of the data. For NCI, models are learned from only 25% of the available training data in every split. Bold font indicates whether single-task or multi-task learning yielded better results

Data Set	KFOIL			
	KTA Scoring		AUC Scoring	
	ST	MT	ST	MT
Alzheimer amine	**91.1** ± 2.4	90.1 ± 2.8	93.9 ± 3.6	**95.6** ± 2.2
Alzheimer toxic	98.1 ± 0.8	**98.3** ± 0.7	95.9 ± 3.4	**96.6** ± 2.0
Alzheimer acetyl	89.8 ± 2.2	**90.0** ± 2.6	92.1± 4.6	**93.8** ± 1.9
Alzheimer memory	82.5 ± 5.0	**86.3** ± 2.7	**86.5** ± 5.8	85.1 ± 4.8
NCGC BJ	68.0 ± 9.3	**69.9** ± 9.9	**72.4**± 10.0	72.2 ± 8.7
NCGC Jurkat	64.7 ± 5.3	**66.2** ± 5.6	69.9 ± 6.0	**71.7** ± 5.6
NCGC Hek293	67.2 ± 7.6	**68.7** ± 7.2	71.0 ± 7.7	**72.0** ± 6.8
NCGC HepG2	71.5 ± 8.7	**74.6** ± 8.8	74.2 ± 8.8	**77.6** ± 8.2
NCGC MRC5	65.3 ± 10.0	**68.8** ± 9.4	67.0 ± 9.6	**69.8** ± 9.3
NCGC SK-N-SH	63.6 ± 7.1	**66.0** ± 7.8	66.4 ± 7.4	**67.2** ± 7.0
MTDP E.coli	61.5 ± 9.3	**62.6** ± 8.0	61.8 ± 8.0	**64.5** ± 7.0
MTDP Human	56.8 ± 9.7	**61.2** ± 10.5	62.5 ± 11.7	**66.7** ± 10.7
MTDP HIV-2	58.8 ± 8.8	**61.7** ± 8.8	61.0 ± 10.1	**63.5** ± 10.1
NCI BT_549 25%	69.7 ± 2.3	**71.1** ± 2.2	70.5± 2.6	**71.8** ± 2.5
NCI HCC_2998 25%	65.3 ± 2.5	**65.8** ± 2.1	64.1 ± 2.5	**67.5** ± 2.6
NCI HS_578T 25%	70.5 ± 2.2	**71.1** ± 2.2	70.3± 2.5	**71.9** ± 2.6
NCI SR 25%	70.3 ± 3.1	**71.2** ± 2.4	69.9 ± 2.9	**71.3** ± 2.4
NCI T_47D 25%	71.1 ± 2.3	**72.2** ± 2.1	70.9 ± 3.0	**72.5** ± 2.5

sets, only 25% of the available training data was used to infer a model to reduce the total computational cost of experiments. Results indicate that multi-task learning provides small but consistent improvements in test set AUC over single-task learning. This is particularly evident on the NCGC, MTDP and NCI data sets, while the result for the Alzheimer data sets is less clear. A likely explanation is that the different tasks in Alzheimer—predicting a compounds amine re-uptake, toxicity, acetyl cholinesterase inhibition, and reversal of memory deficiency—are not as strongly related as for the other domains. Moreover, results confirm the earlier observation that evaluating clause sets by the performance of the corresponding support vector machine (in this case, by its AUC) yields slightly better results than evaluating them by kernel target alignment.

We furthermore investigate how the gains from multi-task learning depend on the amount of training data available. Figure 1 shows average AUC of KFOIL in the single-task and multi-task learning setting for different numbers of training examples on NCI, averaged over the five available tasks. The system has been run with KTA scoring and beam size 5 to achieve maximum computational efficiency for these larger data sets. To obtain stable AUC estimates, the following methodology was used. A 5-fold cross-validation is performed. However, for each fold, 10 models are built on bootstrapped samples of the fold's training set, and their results on the fold's test set are averaged. As in standard cross-validation, this yields one datapoint (AUC result) per fold. Figure 1 also reports one-standard-deviation er-

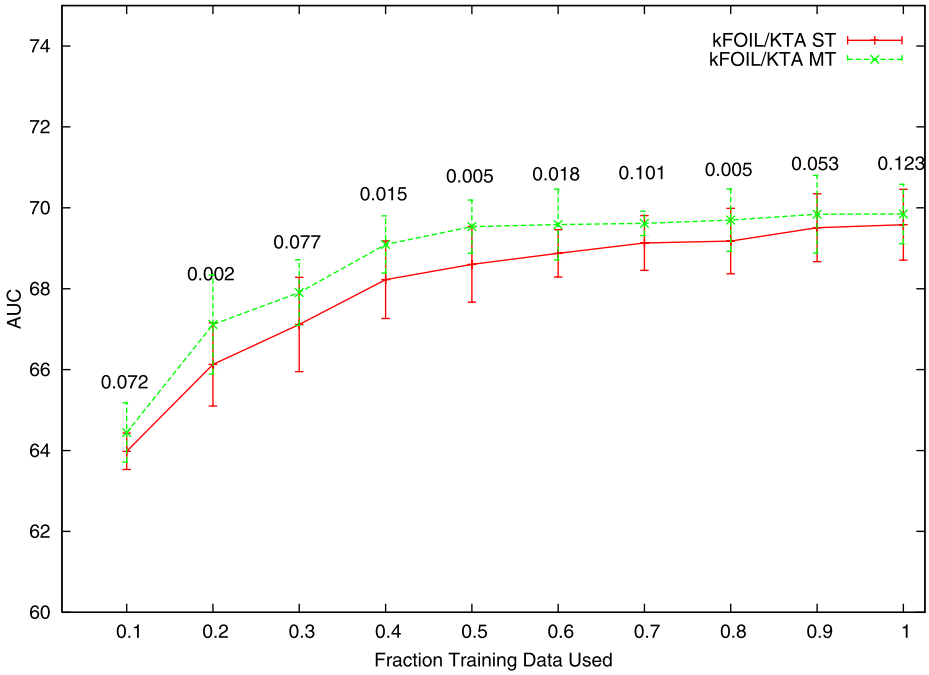


Fig. 1 Learning curve for kFOIL in the NCI domain, with KTA scoring and beam size 5 in a single-task and multi-task learning setting (test set AUC as a function of training set size). Results are averaged over a 5-fold cross-validation, and over individual tasks. For each fold, 10 models are built on bootstrapped samples of the fold's training set and their results on the test set averaged (see text)

ror bars, and for every fraction of training data the significance of the difference between single- and multi-task learning according to a paired two-sided t-test on the fold results. Results again indicate an advantage of multi-task (MT) over single-task (ST) learning. More specifically, MT learning from 50% of the available training data reaches about the same accuracy as ST learning from the whole data set. The difference between MT and ST learning is significant or borderline significant for small training set sizes, but becomes less pronounced if more training data is available.

6.6 Multi-class problems

As a relational multi-class problem, we consider the "University Computer Science Department", or *WebKB*, data set. This data set consists of web pages collected from four computer science departments: Cornell University, University of Texas, University of Washington and University of Wisconsin. Web pages are classified into six categories: `course`, `department`, `faculty`, `research project`, `staff` and `student`. Pages not belonging to any of these classes are assigned to a default `other` class. Table 6 reports class statistics for each of the four Universities. We relied on a relational version of the data set which includes hyperlink and anchor word information (Slattery and Craven 1998). Table 7 reports the predicates we employed in our experimental evaluation.[8] In order to decrease class skew

[8] All predicates where taken from Slattery and Craven (1998) except for `dirs_after_tilde_in_url/2` which was our addition.

Table 6 Class statistics for the "University Computer Science Department" data set

Class	Cornell	Texas	Washington	Wisconsin
course	44	38	77	85
department	1	1	1	1
faculty	34	46	31	42
research project	20	18	21	25
staff	21	3	10	12
student	128	148	126	156
other	619	573	940	946

Table 7 Description of the relational predicates employed for the "University Computer Science Department" data set

Predicate	Description
has_word(page,word)	word is contained in the page text
has_anchor_word(anchor,word)	word is contained in the anchor text
all_words_capitalized(anchor)	all words in the anchor text are capitalized
has_alphanumeric_word(anchor)	the anchor text contains an alphanumeric word
linktopage(page1,page2,anchor)	anchor identifies a link from page1 to page2
dirs_after_tilde_in_url(page,num)	The URL of page contains a tilde followed by num directories

and to obtain simpler and more readable models, we removed the other class in all experiments. In a preprocessing phase, we furthermore selected for each training set the first 50 words and 50 anchor words with highest information gain, and restricted clauses to only contain these informative words.

As outlined in Sect. 3.6, multi-class problems can be cast in the multi-task setting by identifying classes with tasks. We compare a single-task approach (that is, a standard one-vs-rest setup) against multi-task approaches using average single-task scores or multi-class accuracy. Table 8 reports experimental results for these three settings and different scoring functions, in a leave-one-university-out cross-validation (that is, Web pages of one university are classified using a model trained on the remaining three universities). The evaluation measure on the test set is multi-class accuracy. Results confirm the advantage of multi-task learning with respect to single-task, as multi-task scoring functions achieve better results in all cases. It is interesting to note that while multi-task KTA achieves the overall best results, its single-task counterpart performs very badly for the two Universities with the least number of examples, Cornell and Texas. Such performance degradation is due to a very low recall on the student class, which is mostly predicted as course and faculty in the Cornell and Texas cases respectively. Note also that directly using multi-class accuracy as a scoring function does not improve over average accuracy which achieves slightly better overall results. However, the former function tends to learn simpler kernels, with 22.5 clauses on average, with respect to 42.5 learned by average accuracy. KTA also learns quite simple kernels, with 25.25 clauses on average, while achieving the same results as average accuracy.

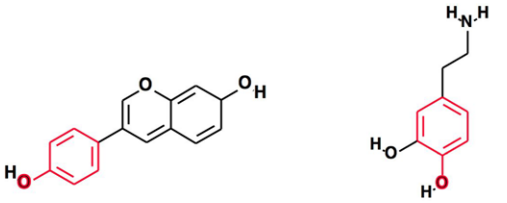
Table 8 Leave-one-university out results for the "University Computer Science Department" data set. Evaluation measure is multi-class accuracy, while scoring measures are single- and multi-task accuracy, AUC and KTA, and multi-class accuracy

Test University	Accuracy		AUC		KTA		Multi-Class Accuracy
	ST	MT	ST	MT	ST	MT	
Cornell	69.8	75.4	77.4	77.4	31.9	75.8	72.6
Texas	74.8	75.2	80.7	81.9	34.6	83.1	81.1
Washington	75.2	74.4	68.8	74.4	75.2	70.3	70.7
Wisconsin	78.2	82.9	78.8	84.1	76.3	80.4	78.8
Micro Average	74.7	77.3	76.5	79.7	56.2	77.5	75.9

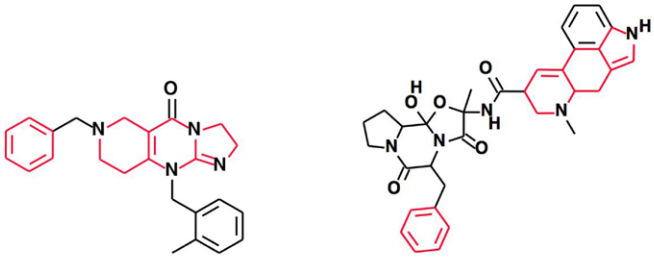
6.7 Interpreting learned models

A major advantage of the proposed methodology for learning relational kernels compared to other relational kernel-based approaches (such as pre-defined kernels for structured data, cf. Gärtner 2003) is that it retains some of the interpretability of its underlying inductive logic programming approach. After training, KFOIL returns a relatively small set of first-order logical clauses that define a similarity measure between examples in the given domain. These clauses are typically easy to read by human experts, especially as they can build on human-supplied background knowledge (such as known functional groups for chemical compounds). Figure 2 shows example clauses learned in the NCTRER, NCI and WebKB domains, and visualizes how these clauses match on examples.

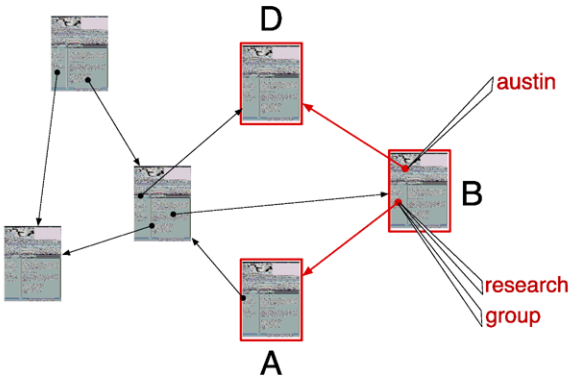
The key part in understanding a final learned model is to understand the similarity function defined by the kernel. The kernel function $k(e_1, e_2)$ is defined by the number of clauses that match both e_1 and e_2, or some non-linear transformation thereof (cf. Sect. 3.3). That is, the kernel counts how many structural features are shared by the two examples. Examples that share a large number of features will be considered similar, and are thus likely to receive the same classification. If clauses are understandable the human experts, the resulting similarity function will be understandable as well as long as the number of features shared between two examples is not overwhelmingly large. Figure 3 shows a histogram representation of the average number of clauses matching an example, and the average number of features shared by a pair of examples. It can be observed that for most pairs of examples the number of shared features is very small, thus it will be easy to manually inspect their similarity. Figure 4 visualizes a learned kernel function on the WebKB data set, for the single-task (left) and multi-task (right) case respectively. In the single-task case, `student` is the positive class, as it leads to the most balanced task. Lighter colors correspond to larger kernel values. Examples are grouped according to classes, and classes ordered so to maximize similarity between neighboring one. Within each class, examples are sorted according to their principal component, in order to cluster together similar examples. In the single-task case, it can be observed that positive examples (bottom left) are roughly grouped together into block-like clusters by the structural features they exhibit. Negative examples have lower kernel values in general, meaning that the predictor tends to rather model the positive class, a common behavior when negative examples come from many possible sources. Anyhow, a slight tendency for negative examples to cluster together on a per-class basis can still be recognized. In the multi-task case, on the other hand, two main differences can be observed: first, the matrix exhibits a coarser grain structure, with larger clusters on average. Indeed, the multi-task predictor tends to learn much smaller sets of clauses, as showed in Table 9.



$$\leftarrow atm(B,o), bd\_atm(B,C,c,-), bd\_atm(C,D,c,=), bd\_atm(C,E,c,-),$$
$$bd\_atm(E,F,c,=), bd\_atm(G,D,c,-), bd\_atm(F,H,I,-)$$



$$\leftarrow three\_linear\_rings(A,B), benzene(A,C,D)$$



$$\leftarrow linktopage(B,A,C), anchor\_word(C,research), anchor\_word(C,group),$$
$$linktopage(B,D,E), anchor\_word(E,austin))$$

Fig. 2 Examples for clauses learned by kFOIL on the NCTRER (*upper*), NCI (*middle*) and WebKB (*lower*) data sets. Additionally, examples on which the clauses match are shown, with the sub-structure defined by the clauses highlighted in red. Note that for NCTRER, only low-level atom/bond structure is given, and the algorithm automatically infers that aromatic rings with a phenol group are relevant for the classification problem at hand. For NCI a library of high-level chemical structures was supplied as background knowledge, such that small clauses can encode relatively complex sub-structures

Second, even if the `student` class is still much more represented, being by far the majority one (see Table 6), other classes are modeled as well, and their clusters are more evident than in the single-task case. However, note that there is a tendency for `staff` and `faculty`

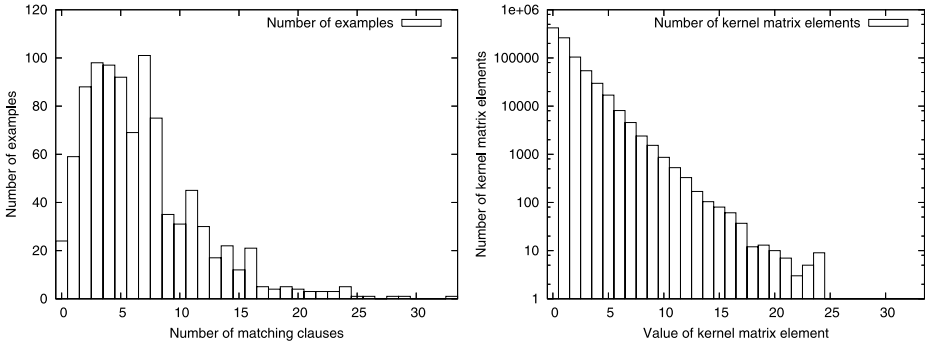


Fig. 3 Histogram representation of the average number of clauses matching an example (*left*), and the average number of features shared between a pair of examples, that is, value of the corresponding kernel matrix element (*right*). Note the logarithmic scale in the second plot. Statistics are collected from NCI-BT_549 (25%), building a model on all of the training data, AUC scoring

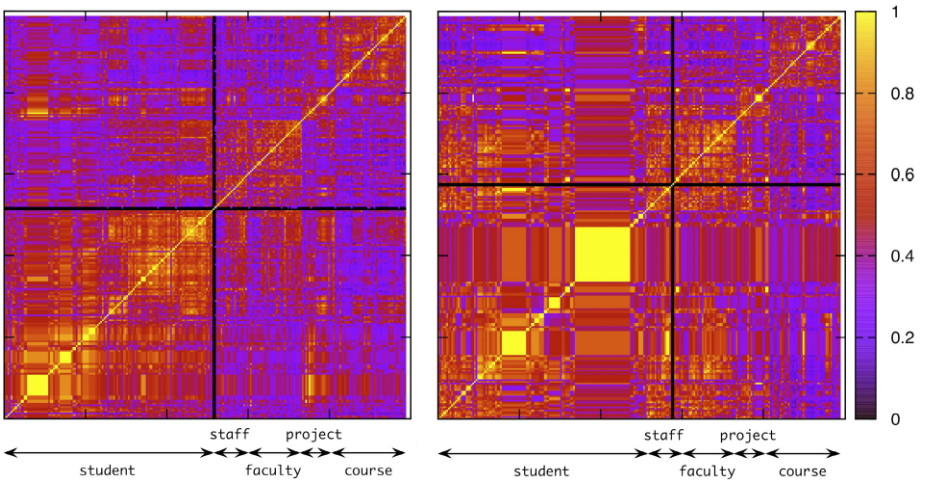


Fig. 4 Visualization of learned kernel functions on the WebKB data set, in a single-task (*left*) and multi-task (*right*) case respectively. Kernel values are shown for webpages at Cornell University for a model trained on the remaining three universities (Washington, Wisconsin, and Texas). In the single-task case, `student` is the positive class as it leads to the most balanced task. Examples are grouped by class, classes are sorted so to maximize similarity between pairwise ones. Examples within each class are sorted by their principal component, in order to cluster together similar ones. Note that there were two "staff" pages from the Cornell CAC (instead of CS department) which did not contain any informative words, and are not matched by any clause, leading to kernel entries of zero. These are visible as *black lines* in the kernel matrix

classes to share a common representation, and to share features with some of the `student` examples. This is quite intuitive given that all such classes represent personal homepages.

Table 9 shows the number of clauses obtained on the Alzheimer, EPA, MTDP, NCI, and WebKB data sets for single-task and multi-task learning. Results show that multi-task learning yields a more compact representation than single-task learning: the set of clauses obtained is significantly smaller than the union of the task-specific clause sets. This indicates that it is possible to infer a clause set (and thus, similarity measure) that generalizes over the

Table 9 Number of clauses obtained for single-task and multi-task learning on the Alzheimer, NCGC, MTDP, NCI, and WebKB data sets (AUC scoring). The clause set for single-task learning is the union of the clause set obtained on the individual tasks (that is, duplicate clauses have been removed). For Alzheimer, NCGC, MTDP, and NCI, results are averaged over a 50 train-test splits as in Table 5. For WebKB, results are averaged over a leave-one-university-out cross-validation as in Table 8

Data Set	Number of Clauses	
	ST	MT
Alzheimer	79.5	34.7
NCGC	195.7	76.0
MTDP	84.0	59.7
NCI	414.5	175.3
WebKB	127.5	37.8

individual tasks. Interpreting such a generalized representation will typically be easier than looking at all task-specific clause sets individually.

6.8 Computational complexity

Computational complexity in kFOIL is dominated by the evaluation of candidate clauses within the greedy top-down refinement search (cf. Algorithm 1). More specifically, for every candidate clause under consideration it has to be determined (1) which examples are covered by the clause and (2) how adding the clause to the current model affects the score. Task (1) consists of running a Prolog query against the current database that holds the description of the examples and the background knowledge. This is a standard task that has to be carried out in ILP systems, and thus constitutes a "computational baseline" in the sense that it is the minimum effort any system has to perform. Task (2) is the additional effort required to score the hybrid statistical-logical model defined by kFOIL, and thus constitutes the computational "overhead" compared to a purely logical approach. Note that the complexity of the second task will strongly depend on the particular scoring function used.

Figure 5 shows the scaling behavior of kFOIL with AUC scoring (beam size 5) and KTA scoring (beam sizes 5 and 20). Complexity is broken down into time spent on Task (1) and Task (2). Results clearly show that KTA scoring scales better than AUC scoring, even taking into account the larger beam size. For KTA scoring coverage calculations clearly dominate overall runtime, and computing kernel target alignment for a candidate clause only constitutes a small overhead. In contrast, for AUC scoring the actual score update strongly dominates the total computational effort. Overall, kFOIL with KTA scoring scales roughly linear in the number of examples, while AUC scoring exhibits clearly non-linear scaling. Note that scoring by classification accuracy has essentially the same complexity as scoring by AUC, as in both cases the full SVM model has to be built.

Additional computational savings are obtained in a multi-task setting: coverage computations only have to be carried out once (as only one clause set is learned). For KTA scoring, where coverage calculations dominate computational cost, multi-task learning thus yields significant computational savings compared to building an individual model for every task.

The (roughly) linear scaling behavior of kFOIL with KTA scoring is surprising, as even the incremental algorithm for computing the alignment of a clause set involves operations which are non-linear in the number of examples (cf. Algorithms 2, 4, and 3). However, note that the relevant factor is the number of *effective* examples, i.e., examples that are mapped

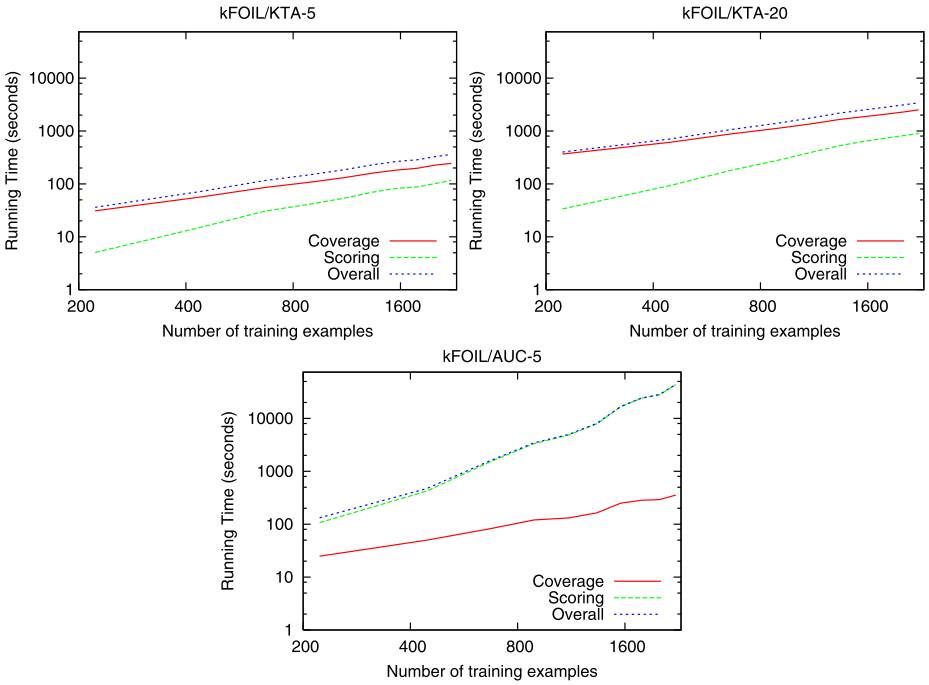


Fig. 5 Runtime in seconds for building a single model on NCI-BT_549 as a function of the number of training instances for different scoring functions and beam sizes. Graphs show (**1**) time spent on determining the coverage of a clause ("Coverage"), (**2**) time spent to determine the score of the combined model given the coverage ("Scoring"), and (**3**) overall time spent in the evaluation of the clause ("Overall"). Results are averaged over 50 random samples for each data set size

to different propositional vectors by the clause set (see Sect. 5). Figure 6, left plot, shows the number of effective examples as a function of the overall number of examples. A least-squares curve fit of the function $f(x) = x^a$ to the (normalized) curve yields $a \approx 0.5187$. Thus, the number of effective examples grows approximately with the square root of the total number of examples, explaining the overall linear scaling behavior observed in Fig. 5.

6.9 Summary of experimental results

As already reported in Landwehr et al. (2006), kFOIL compares favorably against well-known ILP systems and static propositionalization approaches. New results were obtained with more efficient incremental algorithms, kernel target alignment scoring, and in multi-task and multi-class learning settings.

The comparison between AUC and accuracy scoring on the one hand and alignment scoring on the other hand has been shown to be a classical accuracy-efficiency trade-off. Alignment scoring yields slightly lower accuracy, but offers a much better scaling behavior, making the proposed algorithm practical for large-scale relational learning problems. According to our experiments, linear scaling can be expected—a surprising result. The explanation is that the number of *effective* examples that are fed to the support vector machine only grows with the square root of the original number of examples.

Multi-task learning has been shown to offer three key benefits compared to single-task learning. First, our experiments confirm the observation that multi-task learning can offer

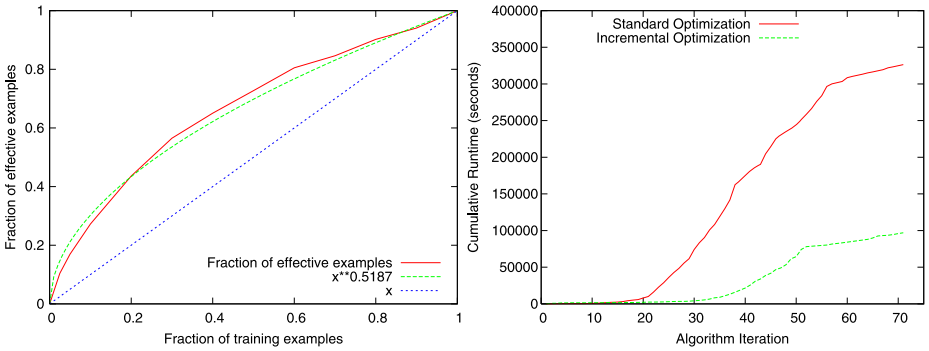


Fig. 6 *Left plot*: number of effective examples as a function of the overall number of examples for KFOIL with KTA scoring and beam size 5. Both axis are normalized to the interval $[0, 1]$, that is, values divided by their corresponding maximum. A least-squares curve fit of the function $f(x) = x^a$ yields a constant $a \approx 0.5187$. Results are averaged over all tasks in NCI and 50 random splits into training (80%) and test (20%) data. *Right plot*: Cumulative runtime of KFOIL training as a function of the algorithm iteration for AUC scoring, with and without incremental optimization of the support vector machine (NCI BT_549 dataset)

advantages in terms of generalization performance (Caruana 1997). Second, the resulting representation of the learned model in terms of a clause set is more compact, as the joint clause set is smaller than the union over the task-specific clause sets induced. Finally, together with alignment scoring multi-task learning offers additional computational benefits, as coverage calculations are shared between tasks.

7 Conclusions

We developed a general framework for statistical relational learning with kernels, and introduced kFOIL as a simple implementation within such framework. We showed how to efficiently learn relational kernels retaining much of the interpretability of ILP systems. Experimental comparisons show the advantages of the proposed approach over ILP systems as well as both static and dynamic propositionalization approaches. Our formulation allows one to naturally handle multi-task learning problems, resulting in a novel structural multi-task learning algorithm. An extensive experimental evaluation proves the advantage of the multi-task learning approach over its single-task counterpart, both in efficiency and effectiveness. We also showed the advantage of multi-task learning in dealing with multi-class classification problems.

kFOIL can be extended in a number of directions. Greedy search, for instance, produces a small but suboptimal set of features, and more complex strategies can be employed, trading efficiency and interpretability for effectiveness. Concerning multi-task learning, while kFOIL currently learns a feature space representation which is common across tasks, it is possible to extend it with task-specific components accounting for the specificity of each task, as is done with hierarchical Bayesian models (see e.g. Deshpande et al. 2007). Finally, it would be interesting to verify if the advantage we observed addressing multi-class classification as a multi-task problem is confirmed with different multi-task learning algorithms.

Acknowledgements We would like to acknowledge support for this work from the Research Foundation Flanders (FWO-Vlaanderen), and the GOA 08/008 project on "Probabilistic Logic Learning".

Appendix: kFOIL incremental algorithm

Algorithm 2 summarizes how relevant information is incrementally updated when scoring a candidate clause. To implement merging examples with the same feature representation (i.e., examples within the same cluster), we keep a mapping of examples to an ID representing their cluster (*E2I*), and a mapping from cluster IDs to the set of examples within this cluster (*I2E*). These mappings, as well as the current kernel matrix M and the examples covered by the candidate clause form the input. The algorithm relies on a local structure of active clusters (*AI2E*). An active cluster is a cluster containing at least one example covered by the candidate clause. *AI2E* maps each active cluster ID to the subset of examples in that cluster which are covered by the clause. Such structure is filled in lines 2–4. Then, the algorithm cycles over active clusters, and determines whether an active cluster is split by the current clause (line 6). This happens when the size of the active cluster is different from the size of the full cluster, implying that some of the examples in the cluster were not covered by the clause, which leads to a different feature space representation. If a split does not occur, the active site ID is simply added to a local set (U) of clusters for which kernel values should be updated (line 7). Otherwise, a split implies getting a new cluster ID (line 9), adding it

Algorithm 2 Procedure to update information needed to score candidate clause

```
1:  procedure AddClause(Ex, I2E, E2I, M)
    Input:
    Ex set of examples covered by the candidate clause
    E2I map from example to feature space id
    I2E map from feature space id to set of examples collapsed into it
    M kernel matrix in lower triangular form
2:      for all i ∈ Ex do                          ▷ group covered examples by feature space id
3:          Insert(AI2E[E2I[i]], i)
4:      end for
5:      for all (id, id2E) ∈ AI2E do
6:          if Size(id2E) = Size(I2E[id]) then     ▷ all id-mapped examples are covered
7:              Insert(U, id)                      ▷ add id to set of ids to update
8:          else                                   ▷ id has to be splitted
9:              newid ← Size(I2E)                  ▷ generate a new id
10:             Insert(U, newid)                   ▷ add newid it to ids to update
11:             Insert(I2E[newid], id2E)           ▷ add newid with covered examples to I2E
12:             for all j ∈ id2E do
13:                 E2I[j] ← newid                 ▷ map example to new id
14:                 Remove(I2E[i], j)              ▷ remove example from old id
15:                 SplitAlphas(id2E, I2E[i])      ▷ split alpha between old and new ids
16:             end for
17:             CopyRow(M, id, newid)              ▷ init new id matrix entries to old id ones
18:             M[Index(newid, newid)] ← M[Index(id, id)]
19:         end if
20:         RemoveFromKTA(id, id2E)                ▷ remove id contribution to KTA
21:     end for
22:     IncrementMatrix(M,U)                       ▷ increment matrix entries of all ids to update
23:     AddToKTA(U)                                ▷ add new id contributions to KTA
24: end procedure
```

Algorithm 3 Procedure for removing obsolete contributions from KTA

1: **procedure** REMOVEFROMKTA($id, id2E, I2E, I2E_{old}, KTA$)
Input:
$id, id2E$ id to be (partially) removed from KTA, and examples mapped into it
$I2E, I2E_{old}$ feature space id to examples maps, current and before adding clause
KTA kernel-target, kernel-kernel and target-target Frobenius norms
2: **for all** $ti \in \{-1, 1\}$ **do**
3: $tiS \leftarrow$ SIZE($id2E, ti$) ▷ recover size of examples with target ti
4: **for all** $(idj, idj2E) \in I2E_{old}$ **do** ▷ iterate over original ids
5: $k \leftarrow$ KERNEL(id, idj)
6: **for all** $tj \in \{-1, 1\}$ **do**
7: $T \leftarrow ti \cdot tj$
8: $tjS \leftarrow$ SIZE($idj2E, tj$)
9: **if** $id = idj$ and $ti = tj$ **then** ▷ Treat diagonal entry differently
10: $S \leftarrow tiS(2 \cdot tjS - tiS)$
11: **else if** $idj < id$ or $tj < ti$ **then** ▷ Only remove left idj size
12: $S \leftarrow 2 \cdot tiS \cdot$ SIZE($I2E[idj], tj$)
13: **else**
14: $S \leftarrow 2 \cdot tiS \cdot tjS$
15: **end if**
16: DECREASEKTA($k \cdot T \cdot S, k \cdot k \cdot S, T \cdot T \cdot S$)
17: **end for**
18: **end for**
19: **end for**
20: **end procedure**

to the set U of clusters to update (line 10) and updating the mapping $I2E$ (lines 11 and 14) and $E2I$ (line 13). Kernel values from the old cluster ID should be copied on a new row of the kernel matrix (line 17), and a new diagonal entry should be set equal to that of the old id (line 18). Finally, kernel values corresponding to a pair of cluster IDs which are both covered by the new clause (U) are incremented by 1 to account for the contribution of the new clause (line 22).[9]

Depending on the scoring function employed, additional updates are also performed: if scoring involves solving a minimization problem (Case 3), computational savings can be obtained by restarting from a previously obtained solution (see Sect. 5.3). Note that if a split occurs, the corresponding alpha value has to be distributed among splitted IDs (line 15), so that the starting point is consistent with constraints.

On the other hand, if scoring involves KTA computation, the algorithm simply updates the contribution of covered examples to the previously computed KTA. This is done by updating the three Frobenius norms from which KTA is computed (see (10)). For each cluster ID, the contribution of the corresponding examples which are covered by the clause is first removed from the norms (line 20, see Algorithm 3) and then replaced with the contribution according to their updated kernel value (line 23, see Algorithm 4).

Consider Algorithm 3, which removes contribution of cluster id with corresponding examples $id2E$, i.e., the examples which are covered by the candidate clause, from the norms.

[9]This is assuming a linear kernel; combinations with polynomial or Gaussian kernels can be easily implemented given the linear kernel matrix.

Algorithm 4 Procedure for adding new contributions to KTA

```
1:  procedure AddToKTA(Ids, I2E, KTA)
    Input:
    Ids ids to be added to KTA
    I2E map from feature space id to corresponding set of examples
    KTA kernel-target, kernel-kernel and target-target Frobenius norms
2:      for all id ∈ Ids do
3:          for all ti ∈ {−1, 1} do
4:              tiS ← Size(I2E[id], ti)
5:              for all (idj, idj2E) ∈ I2E do
6:                  k ← Kernel(id, idj)
7:                  for all tj ∈ {−1, 1} do
8:                      T ← ti · tj
9:                      S ← tiS · Size(idj2E, tj)
10:                     if !Find(Ids, idj) then
11:                         S ← S · 2
12:                     end if
13:                     IncreaseKTA(k · T · S, k · k · S, T · T · S)
14:                 end for
15:             end for
16:         end for
17:     end for
18: end procedure
```

Additional inputs to the algorithm, which were omitted from the call for the sake of readability, are the current mapping from cluster IDs to examples (*I2E*), and the original mapping ($I2E_{old}$) before adding the clause, as well as the current three Frobenius norms (in the *KTA* structure). Removing the contribution of *id* from the norms amounts at removing the values of the corresponding row (and column) in the kernel-target, kernel-kernel and target-target matrices (see (10)). In two out of three cases, we have to distinguish between positive and negative examples. Let tiS_+ and tiS_- be the positive and negative examples in *id2E* (line 3). By iterating over cluster identifiers *idj* (line 4), the algorithm scans the row (and column) to be removed. First it computes kernel $kij = K(idi, idj)$ (line 5), and target $T = ti \cdot tj$ (line 7) matrix entries, the latter for both positive and negative case of *idj*. Then, it computes the number of times such matrix entries should be counted (remember that a matrix entry corresponds to a cluster of examples with same feature representation). When multiple IDs are affected by a clause, care must be taken in not removing the contribution of the same set of examples multiple times. This is accomplished by considering both the original size (*tjS*, line 8) of cluster *idj*, i.e., before adding the clause, and the current size (Size(*I2E*[*idj*], *tj*), see line 12), which can be smaller if a split on *idj* has occurred before the current call of the algorithm. Three cases should be considered. (1) diagonal entry (line 10): here $idj = id$, *tiS* is the amount to move (examples covered by the clause) and *tjS* the original size. If we split the original size in moved (*tiS*) and left (*tlS*) parts, we have $(tlS + tiS)^2 = tlS^2 + tiS^2 + 2 \cdot tlS \cdot tiS$, and as only the contribution of the moved examples should be removed from the norms, we obtain $tiS^2 + 2 \cdot tlS \cdot tiS = tiS(tiS + 2(tjS - tiS)) = tiS(2 \cdot tjS - tiS)$. (2) part of contribution already removed (line 12), either by previous calls ($idj < id$) or previous iterations within the current call ($tj < ti$): only consider the current size for *idj*. (3) novel removal (line 14): remove the entire original size for *idj*. Finally, the correct amount is removed from the free

Frobenius norms (line 16), combining the values computed in the previous steps. In a similar but simpler fashion, Algorithm 4 adds the contribution of the covered examples to the norms. Note that the procedure ADDCLAUSE is called for every candidate clause to update the model by the contribution of that clause. After the candidate clause has been scored, a simple procedure REMOVECLAUSE (not shown) is called to remove its contribution before evaluating an alternative refinement, basically by decrementing the kernel matrix and restoring previous mappings and either alphas or Frobenius norms depending on the scoring function employed.

References

Argyriou, A., Hauser, R., Micchelli, C. A., & Pontil, M. (2006). A DC-programming algorithm for kernel selection. In *Proceedings of the 23rd international conference of machine learning (ICML-2006)* (pp. 41–48), Pittsburgh, PA, USA.

Argyriou, A., Evgeniou, T., & Pontil, M. (2007). Multi-task feature learning. In *Advances in neural information processing systems 19* (pp. 41–48). Cambridge: MIT Press.

Bartlett, P. L., Jordan, M. I., & McAuliffe, J. D. (2006). Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, *101*, 138–156.

Ben-David, S., Eiron, N., & Simon, H. U. (2002). Limitations of learning via embeddings in Euclidean half spaces. *Journal of Machine Learning Research*, *3*, 441–461.

Bengio, Y., Delalleau, O., & Roux, N. L. (2005). The curse of highly variable functions for local kernel machines. In *Advances in neural information processing systems 18*. Cambridge: MIT Press.

Blockeel, H., De Raedt, L., & Ramon, J. (1998). Top-down induction of clustering trees. In *Proceeding of the 15th international conference on machine learning*, Madison, Wisconsin, USA.

Blockeel, H., Dzeroski, S., Kompare, B., Kramer, S., Pfahringer, B., & Laer, W. (2004). Experiments in predicting biodegradability. *Applied Artificial Intelligence*, *18*(2), 157–181.

Bordes, A., Ertekin, S., Weston, J., & Bottou, L. (2005). Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, *6*, 1579–1619.

Caponnetto, A., Micchelli, C., Pontil, M., & Ying, Y. (2008). Universal kernels for multi-task learning. *Journal of Machine Learning Research*.

Caruana, R. (1997). Multitask learning. *Machine Learning*, *28*(1), 41–75.

Chapelle, O., Vapnik, V., Bousquet, O., & Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning*, *46*(1–3), 131–159.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*, 1–25.

Cristianini, N., Shawe-Taylor, J., Elisseef, A., & Kandola, J. (2001). On kernel-target alignment. In *Advances in neural information processing systems 14*. Cambridge: MIT Press.

Cucker, F., & Smale, S. (2002). On the mathematical foundations of learning. *Bulletin (New Series) of the American Mathematical Society*, *39*(1), 1–49.

Cumby, C. M., & Roth, D. (2003). On kernel methods for relational learning. In *Proceedings of the twentieth international conference on machine learning* (pp. 107–114), Washington, DC, USA.

Datta, P., & Kibler, D. F. (1993). Concept sharing: a means to improve multi-concept learning. In *Proceedings of the 10th international conference on machine learning*, Amherst, MA, USA.

Davis, J., Burnside, E., de Castro Dutra, I., Page, D., & Costa, V. S. (2005). An integrated approach to learning Bayesian networks of rules. In *Lecture notes in computer science: Vol. 3720. Machine learning, 16th European conference* (pp. 84–95), Porto, Portugal. Berlin: Springer.

De Raedt, L. (2008). *Logical and relational learning*. Berlin: Springer.

De Raedt, L., & Ramon, J. (2004). Condensed representations for inductive logic programming. In *Proceedings of the 9th international conference on the principles of knowledge representation and reasoning*.

De Raedt, L., Lavrac, N., & Dzeroski, S. (1993). Multiple predicate learning. In *Proceedings of the 13th international joint conference on artificial intelligence* (pp. 1037–1043), Chambery, France.

De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (2008). *Lecture notes in computer science: Vol. 4911. Probabilistic inductive logic programming—theory and applications*. Berlin: Springer.

Dehaspe, L., Toivonen, H., & King, R. (1998). Finding frequent substructures in chemical compounds. In *Proceedings of the 4th international conference on knowledge discovery and data mining*.

Deshpande, A., Milch, B., Zettlemoyer, L., & Kaelbling, L. (2007). Learning probabilistic relational dynamics for multiple tasks. In *Proceedings of the 23rd conference on uncertainty in artificial intelligence (UAI-07)* (pp. 83–92).

Evgeniou, T., Micchelli, C. A., & Pontil, M. (2005). Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, *6*, 615–637.

Fang, H., Tong, W., Shi, L., Blair, R., Perkins, R., Branham, W., Hass, B., Xie, Q., Dial, S., Moland, C., & 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.

Frasconi, P., Passerini, A., Muggleton, S., & Lodhi, H. (2005). Declarative kernels. In Kramer, S., & Pfahringer, B. (Eds.), *Proceedings of the 15th international conference on inductive logic programming, late-breaking papers* (pp. 17–19).

Frasconi, P., Jaeger, M., & Passerini, A. (2008). Feature discovery with type extension trees. In *Lecture notes in computer science: Vol. 5194*. *ILP 2008: Proceedings of the 18th international conference*. Berlin: Springer.

Freund, Y., & Schapire, R. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, *37*(3), 277–296.

Gärtner, T. (2003). A survey of kernels for structured data. *SIGKDD Explorations*, *5*(1), 49–58.

Gärtner, T., Lloyd, J., & Flach, P. (2004). Kernels and distances for structured data. *Machine Learning*, *57*(3), 205–232.

Getoor, L., & Taskar, B. (2007). *Introduction to statistical relational learning*. Cambridge: MIT Press.

Höffgen, K.-U., Simon, H.-U., & van Horn, K. S. (1995). Robust trainability of single neurons. *Journal of Computer and System Sciences*, *50*(1), 114–125.

Jebara, T. (2004). Multi-task feature and kernel selection for SVMs. In *Proceedings of the 21st international conference on machine learning*, Banff, Alberta, Canada.

Joachims, T. (1999). Making large-scale support vector machine learning practical. In *Advances in kernel methods: support vector learning* (pp. 169–184). Cambridge: MIT Press.

Karalič, A., & Bratko, I. (1997). First order regression. *Machine Learning*, *26*(2–3), 147–176.

Kersting, K., & De Raedt, L. (2007). Bayesian logic programming: theory and tools. In Getoor, L. & Taskar, B. (Eds.), *Introduction to statistical relational learning*. Cambridge: MIT Press.

Khan, K., Muggleton, S., & Parson, R. (1998). Repeat learning using predicate invention. In *Lecture notes in computer science: Vol. 1446*. *Inductive logic programming, 8th international workshop, Proceedings* (pp. 165–174), Madison, Wisconsin, USA. Berlin: Springer.

Kimeldorf, G. S., & Wahba, G. (1970). A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, *41*, 495–502.

King, R., Srinivasan, A., & 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., & Horváth, T. (2001). Distance based approaches to relational learning and clustering. In *Relational data mining* (pp. 213–230). Berlin: Springer.

Kok, S., & Domingos, P. (2005). Learning the structure of Markov logic networks. In *Proceedings of the 22nd international conference on machine learning* (pp. 441–448), Bonn, Germany. New York: ACM.

Kramer, S., & De Raedt, L. (2001). Feature construction with version spaces for biochemical applications. In *Proceedings of the 18th international conference on machine learning* (pp. 258–265). San Mateo: Morgan Kaufmann.

Lanckriet, G. R. G., Cristianini, N., Bartlett, P., Ghaoui, L. E., & Jordan, M. I. (2004). Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, *5*, 27–72.

Landwehr, N., Kersting, K., & De Raedt, L. (2005). nFOIL: integrating naive Bayes and FOIL. In *Proceedings of the 20th national conference on artificial intelligence* (pp. 795–800), Pittsburgh, Pennsylvania, USA.

Landwehr, N., Passerini, A., De Raedt, L., & Frasconi, P. (2006). kFOIL: Learning simple relational kernels. In *Proceedings of the 21st national conference on artificial intelligence*, July 16–20, 2006, Boston, Massachusetts, USA.

Lavrac, N., & Dzeroski, S. (1994). *ILP: techniques and application*. Chichester: Ellis Horwood.

Leslie, C. S., Eskin, E., & Noble, W. S. (2002). The spectrum kernel: a string kernel for SVM protein classification. In *Pacific symposium on biocomputing* (pp. 566–575), Lihue, Hawaii, USA.

Lloyd, J. W. (1987). *Foundations of logic programming* (2nd extended ed.). Berlin: Springer.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *The Journal of Machine Learning Research*, *2*, 419–444.

Micchelli, C. A., & Pontil, M. (2005). Learning the kernel function via regularization. *Journal of Machine Learning Research*, *6*, 1099–1125.

Micchelli, C., Xu, Y., & Zhang, H. (2006). Universal kernels. *The Journal of Machine Learning Research*, *7*, 2651–2667.

Muggleton, S. (2000). Learning stochastic logic programs. In Getoor, L. & Jensen, D. (Eds.), *Proceedings of the AAAI2000 workshop on learning statistical models from relational data*.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: theory and methods. *Journal of Logic Programming*, *19/20*, 629–679.

Muggleton, S., Amini, A., & Sternberg, M. (2005). Support vector inductive logic programming. In *Lecture notes in computer science: Vol. 3735. Discovery science, 8th international conference, Proceedings* (pp. 163–175), Singapore. Berlin: Springer.

Obozinski, G., Taskar, B., & Jordan, M. (June, 2006). *Multi-task feature selection*. Technical report, Dept. of Statistics, UC Berkeley.

Ong, C. S., Smola, A. J., & Williamson, R. C. (2005). Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, *6*, 1043–1071.

Passerini, A., Frasconi, P., & De Raedt, L. (2006). Kernels on prolog proof trees: statistical learning in the ILP setting. In *Probabilistic, logical and relational learning—towards a synthesis*.

Platt, J. (1999). Sequential minimal optimization: a fast algorithm for training support vector machines. In *Advances in kernel methods: support vector learning* (pp. 185–208).

Poggio, T., & Smale, S. (2003). The mathematics of learning: dealing with data. *Notices of the American Mathematical Society*, *50*(5), 537–544.

Popescul, A., Ungar, L., Lawrence, S., & Pennock, D. (2003). Statistical relational learning for document mining. In *Proceedings of the 3rd IEEE international conference on data mining* (pp. 275–282).

Provost, F., Fawcett, T., & Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In *Proceeding of the 15th international conference on machine learning*, Madison, Wisconsin, USA.

Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.

Rakotomamonjy, A. (2004). Optimizing area under ROC curve with SVMs. In *Workshop on ROC analysis in AI at the 15th European conference on artificial intelligence*, Pisa, Italy.

Ramon, J. (2002). *Clustering and instance based learning in first order logic*. Ph.D. thesis, Katholieke Universiteit Leuven, Belgium.

Ramon, J., & Bruynooghe, M. (1998). A framework for defining distances between first-order logic objects. In *Lecture notes in computer science: Vol. 1446. Inductive logic programming, 8th international workshop, Proceedings* (pp. 271–280), Madison, Wisconsin, USA. Berlin: Springer.

Reid, M. D. (2004). Improving rule evaluation using multitask learning. In *Lecture notes in artificial intelligence: Vol. 3194. Inductive logic programming, 14th international conference, Proceedings*, Porto, Portugal. Berlin: Springer.

Rückert, U., & Kramer, S. (2007). Margin-based first-order rule learning. *Machine Learning*, *70*(2–3), 189–206.

Saigo, H., Nowozin, S., Kadowaki, T., Kudo, T., & Tsuda, K. (2009). gBoost: a mathematical programming approach to graph classification and regression. *Machine Learning*, *75*(1), 69–89.

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: primal estimated sub-GrAdient SOlver for SVM. *Proceedings of the 24th international conference on machine learning* (pp. 807–814), Corvallis, Oregon, USA.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Slattery, S., & Craven, M. (1998). Combining statistical and relational methods for learning in hypertext domains. In *Lecture notes in computer science.: Vol. 1446. Inductive logic programming, 8th international workshop, Proceedings*, Madison, Wisconsin, USA. Berlin: Springer.

Srinivasan, A., Muggleton, S., King, R., & Sternberg, M. (1996). Theories for mutagenicity: a study of first-order and feature-based induction. *Artificial Intelligence*, *85*, 277–299.

Srinivasan, A., King, R. D., & Bristol, D. (1999). An assessment of ILP-assisted models for toxicology and the PTE-3 experiment. In *Lecture notes in computer science: Vol. 1634. Inductive logic programming, 9th international workshop, ILP-99, Proceedings*, Bled, Slovenia, June 24–27, 1999. Berlin: Springer.

Steck, H. (2007). Hinge rank loss and the area under the ROC curve. In *Proceedings of the 18th European conference on machine learning*, Warsaw, Poland.

Swamidass, S. J., Chen, J., Bruand, J., Phung, P., Ralaivola, L., & Baldi, P. (2005). Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, *21*(1), 359–368.

Wachman, G., & Khardon, R. (2007). Learning from interpretations: a rooted kernel for ordered hypergraphs. In *Proceedings of the 24th international conference on machine learning*, Corvallis, Oregon, USA.

Weston, J., Schölkopf, B., Eskin, E., Leslie, C., & Noble, W. (2003). Dealing with large diagonals in kernel matrices. *Annals of the Institute of Statistical Mathematics*, *55*(2), 391–408.

Wheeler, D. L. L., Barrett, T., Benson, D. A. A., Bryant, S. H. H., Canese, K., Chetvernin, V., Church, D. M. M., Dicuccio, M., Edgar, R., Federhen, S., Feolo, M., Geer, L. Y. Y., Helmberg, W., Kapustin, Y., Khovayko, O., Landsman, D., Lipman, D. J. J., Madden, T. L. L., Maglott, D. R. R., Miller, V., Ostell, J., Pruitt, K. D. D., Schuler, G. D. D., Shumway, M., Sequeira, E., Sherry, S. T. T., Sirotkin, K., Souvorov, A., Starchenko, G., Tatusov, R. L. L., Tatusova, T. A. A., Wagner, L., & Yaschenko, E. (2008). Database resources of the national center for biotechnology information. *Nucleic Acids Research*.