Declarative Kernels

Paolo Frasconi[†] **Andrea Passerini**[†] **Stephen Muggleton**[‡] **Huma Lodhi**[‡]

[†] Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze
[‡] Department of Computing, Imperial College London

Abstract

We introduce a declarative approach to kernel design based on background knowledge expressed in the form of logic programs. The theoretical foundation of declarative kernels is mereotopology, a general theory for studying parts and wholes and for defining topological relations among parts. Declarative kernels can be used to specify a broad class of kernels over relational data and represent a step towards bridging statistical learning and inductive logic programming. The flexibility and the effectiveness of declarative kernels is demonstrated in a number of real world problems.

1 Introduction

Kernels are widely used in machine learning to represent the similarity between instances. Choosing an appropriate kernel (or an appropriate class of kernels) is essentially the main decision affected by the available domain knowledge and may carry paramount consequences on the success of an application. In spite of these considerations, kernels design is very often carried out in a rather ad-hoc manner because of the lack of *systematic* and *flexible* tools for aiding the design task. One difficulty is perhaps that real-valued functions are inappropriate as a knowledge representation language. Declarative languages such as first order logic offer a better alternative. Indeed, symbolic approaches to machine learning such as inductive logic programming (ILP) [12] heavily rely on first order representations of domain knowledge. Flexible tools for exploiting knowledge that has been formalized in a logical language, however, are not usually available when using statistical learning and kernel methods. In this paper, we propose a framework that attempts to fill in the gap.

During the last few years, several approaches have been proposed for defining kernels on discrete data structures (see [5] for a review) and higher-order logic individuals [4]. These kernels, however, operate on *extensional* instances and not on *intensional* knowledge. Other researchers have proposed ideas for incorporating background knowledge into kernel machines. For example, Fung *et al.*'s method [3] incorporates partial knowledge about the target function in the learning process associated with support vector machines. Unfortunately, such methods are not sufficiently flexible to exploit expressive knowledge representation languages such as first-order logic. Propositionalization is a rather different solution that transforms a relational problem into one that can be solved by an attribute-value learner by mapping relations into a fixed-size set of features [11]. Although it is known that in many practical applications propositionalization works well, its flexibility is generally limited. A remarkable exception is the method proposed in [2] that uses description logics to specify features associated with a kernel.

In this paper we aim to introduce a general logical framework that allows us to specify a broad class of kernels and to provide a simple interface for the incorporation of relational background knowledge. Our main focus is not to propose another new kernel on relational data but rather to provide the foundations for a programming environment that incorporates kernel-based learning. Within this environment, one may write programs that are translated into kernel functions over logical representations of *both* data and knowledge. We assume that instances are well defined individuals represented by sets of ground facts and that background knowledge is available in the form of first-order predicates. Learning in this setting is traditionally a prerogative of ILP but in this paper we show that statistical learning with kernels is possible as well.

We propose to start from a specific setting based on two special and germane sets of relations for reasoning about *parts* and *places*. The parthood relation has been formally investigated by logicians and philosophers for almost a century since the early work of Leśniewski, followed by Leonard & Goodman's calculus of individuals (see [1] and references therein). The axiomatic theory of parts is referred to as *mereology* (from the Greek $\mu\epsilon\rho o\zeta$, "part"). This theory already brings some connections to representational issues in learning since decomposition into parts is central to the study of several kernels on discrete structures (e.g. convolution kernels [8]). The theory can be enriched with additional topological predicates and axioms aiming to describe wholeness. As pointed out by Varzi [1], topology is much needed because "mereological reasoning by itself cannot do justice to the notion of a whole (a one-piece, self-connected whole, such as a stone or a whistle, as opposed to a scattered entity made up of several disconnected parts, such as a broken glass, an archipelago, or the sum of two distinct cats)." These ideas are also relevant to learning since pure decompositional approaches may have limited representational power (e.g. bag-of-words representation of documents or representations of proteins as sets of k-mers may destroy potentially useful information). For these reasons, a mereotopological starting point for the definition of kernel functions may have a promising potential.

2 The logical framework

In the following, we base our discussion on standard first-order logic and when discussing programs we implicitly refer to the Prolog programming languages [16] that support predicate declaration and first-order deductive inference via resolution.

2.1 Ontology

Data and domain theory are represented by means of a program in which each instance consists of a set of ground facts and background knowledge consists of a set of first order predicates. We denote by $\mathcal{U}$ the universe of discourse, consisting of a set of objects, and by $\mathcal{X} \subset \mathcal{U}$ the instance space.

To enrich the expressivity of the representation, we can enforce a type discipline on objects by introducing a finite set of monadic type names $\mathcal{T}$ where each type $\tau \in \mathcal{T}$ defines a subset of $\mathcal{U}$. We denote by : the relation between objects and their types, i.e. for $x \in \mathcal{U}$ we write $x : \tau$ to denote that x has type τ. A special case is when $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ is a partition of $\mathcal{U}$. In this case $\mathcal{T}$ can be viewed as an equivalence relation $=_T$ as follows: $\forall x, y \in \mathcal{U}\ x =_T y$ iff $\exists \tau_i \in \mathcal{T} s.t. (x : \tau_i \Leftrightarrow y : \tau_i)$. Another interesting situation is when type names are hierarchically organized in a partial order $\prec_T \subset \mathcal{T} \times \mathcal{T}$, with $\sigma \prec_T \tau$ meaning that σ *is a* τ (e.g. dog $\prec_T$ animal). Numerical and categorical attributes can be attached to objects. We denote by $a[x]$ the value taken on by attribute a in object x. As common in many object-oriented models, we maintain that objects of the same type share the same set of attributes and that subtyping adds attributes to the base type.

The ontology is completed by a set of facts and rules representing instances of parthood and connection relations between objects. For this purpose, we introduce two special predicates in the ontology: $\preceq_P$ and C, with the following intended meaning. For any two objects x

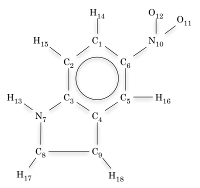


Figure 1: Molecule used to illustrate the examples.

and y, $x \preceq_P y$ if x is declared to be a part of y. Similarly, $C(x, y)$ if x is declared to be connected to y. These predicates should obey general rules that have been the subject of many investigations (as pointed out in the introduction) and that we briefly summarize below.

2.2 Mereotopology

In the context of knowledge representation, it is widely accepted that $\preceq_P$ should be a partial order, i.e. $\forall x, y, z \in \mathcal{U}$

$$x \preceq_P x \tag{P1}$$

$$x \preceq_P y \wedge y \preceq_P x \Rightarrow y =_P x \tag{P2}$$

$$x \preceq_P y \wedge y \preceq_P z \Rightarrow x \preceq_P z \tag{P3}$$

The theory defined by the above axioms is referred to as *ground mereology*. Interestingly, the above theory immediately provides us with a natural identity predicate $=_P$ that may be used as a basic elementary operator for comparing parts. Additional useful relations are supported by the theory, in particular

$$x \prec_P y \text{ iff } x \preceq_P y \wedge \neg y \preceq_P x \qquad \text{proper part} \tag{1}$$

$$O(x, y) \text{ iff } \exists z.(z \preceq_P x \wedge z \preceq_P y) \qquad \text{overlap} \tag{2}$$

$$U(x, y) \text{ iff } \exists z.(x \preceq_P z \wedge y \preceq_P z) \qquad \text{underlap} \tag{3}$$

The *supplementation* axiom, if added to the theory, supports the notion of extensionality:

$$\forall z.(z \preceq_P x \Rightarrow O(z, y)) \Rightarrow x \preceq_P y. \tag{P4}$$

Following [1], the following axioms characterize topology and its link to mereology:

$$C(x, x) \tag{C1}$$

$$C(x, y) \Rightarrow C(y, x) \tag{C2}$$

$$x \preceq_P y \Rightarrow \forall z.(C(z, x) \Rightarrow C(z, y)) \tag{C3}$$

Additional useful relations are supported by the theory, in particular

$$EC(x, y) \text{ iff } C(x, y) \wedge \neg O(x, y) \qquad \text{external connection} \tag{4}$$

In our framework, mereotopology is used to enrich the given ontology in the hope that it will generate further instances of the parthood and connection relations that will be *useful* for learning. It may also serve the purpose of checking the correctness of the declared parts and connections in the ontology. When used for generating new instances of mereotopological relations, the theory should be used wisely or it may give raise to an explosion of uninteresting parts and connections. Thus, depending on the application domain, axioms can be selectively omitted — for example (P4) will be typically avoided. As an extreme case, the whole set of axioms can be omitted.

Example 1. *Suppose x is the benzene ring in Fig. 1. Suppose that every atom of the ring (C_1–C_6) is declared to be a part of x in the ontology. Adding the supplementation axiom (P4) to the theory entails that any subset of atoms (e.g. C_1, C_3, C_4) is also a part of x. While this appears to be correct in the common interpretation, it leads to comparing an exponentially growing number of sub-parts that are uninteresting when seen as features.*

Example 2. *With reference to Fig. 1, suppose that the only pairs declared to be connected in the ontology are bonded atoms. Then we can infer from axiom (C3) that the nitro group consisting of atoms* N_{10}*,* O_{11}*, and* O_{12} *is connected to the benzene ring. The connection is external since it does not involve sharing of parts.*

2.3 Mereotopological relations (MRs)

Several MRs can be introduced to characterize an instance x, for example:

i) the proper parts of x: $\mathcal{R}_P(x) = \{y : y \prec_P x\}$;
ii) the connected proper parts of x: $\mathcal{R}_C(x) = \{(y,z) : y \prec_P x \wedge z \prec_P x \wedge C(y,z)\}$;
iii) the overlapping parts in x, along with their common proper parts: $\mathcal{R}_I(x) = \{(y,z,w) : y \neq z \wedge y \prec_P x \wedge z \prec_P x \wedge w \prec_P y \wedge w \prec_P z\}$;
iv) the externally connected parts in x along with the associated linking terminals: $\mathcal{R}_L(x) = \{(y,z,u,v) : z \prec_P x \wedge y \prec_P x \wedge \neg O(z,y) \wedge u \prec_P z \wedge v \prec_P y \wedge C(u,v)\}$.

Additional MRs can be defined if necessary. We denote by $\mathcal{M}$ the set of declared MRs. As detailed below, a declarative kernel compares two instances by comparing the corresponding MRs, so adding relations to $\mathcal{M}$ plays a crucial role in shaping the feature space.

Example 3. *With reference to Fig. 1, if we denote by x the molecule, by y the benzene ring and by z the 5-ring, then (y,z,C_3) and (y,z,C_4) are both in $\mathcal{R}_I(x)$. Similarly, denoting by v the nitro group, (y,v,C_6,N_{10}) is in $\mathcal{R}_L(x)$.*

3 Declarative kernels

The definition of the kernel between two instances is built up from simpler kernel functions on mereological relations.

3.1 The contribution of parts

The kernel on parts, denoted K_P, is naturally defined as the *set kernel* between the sets of proper parts:

$$K_P(x,x') = \sum_{y \in \mathcal{P}(x)} \sum_{y' \in \mathcal{P}(x')} k_P(y,y') \tag{5}$$

where k_P denotes a kernel function on parts, which is defined recursively using K_P. The precise definition of k_P depends on the organization of types, as discussed below.

Flat type system. Suppose types can be viewed as an equivalence relation ($\mathcal{T}$ is a partition of $\mathcal{U}$). In this case

$$k_P(y,y') = \begin{cases} \iota(y,y') & \text{if } y =_T y' \text{ and } y,y' \text{ are atomic objects;} \\ K_P(y,y') + \iota_(y,y') & \text{if } y =_T y' \text{ and } y,y' \text{ are non atomic objects;} \\ 0 & \text{otherwise (i.e. } y \neq_T y'\text{).} \end{cases} \tag{6}$$

In the above definition, $\iota(y,y')$ is a kernel function between objects that only depends on the attributes of y and y' (an not on their parts). The available attributes will be characterized by the type of y and y'. If there are no attributes, then $\iota(y,y') = 1$. Otherwise:

$$\iota(y,y') = \sum_{\ell} \kappa_\ell(a_\ell[y], a_\ell[y']) \tag{7}$$

where $a_\ell[y]$ is the value of attribute a_ℓ in y and the sum extends to the attributes associated with the type of y (and y'). κ_ℓ is a kernel on attribute a_ℓ, (e.g. $\kappa_\ell(r,s) = r \cdot s$ if a_ℓ is a numerical attribute). As an alternative, the sum between K_P and ι in Eq. 6 can be replaced by a product.

Hierarchical type system. In this case the test for type equality in Eq. 6 can be replaced by a more relaxed test on type compatibility. In particular, if $y : \tau$, $y' : \tau'$ and there exists a least general supertype $\sigma : \tau \prec_T \sigma, \tau' \prec_T \sigma$, then we may type cast y and y' to σ and evaluate κ on the generalized objects $\sigma(y)$ and $\sigma(y')$. In this case the sum in Eq. 7 extends to the subset of attributes for type τ that are also present in type σ.

3.2 The contribution of other MRs

The kernel on connected parts compares the sets of objects $\mathcal{R}_C(x)$ and $\mathcal{R}_C(x')$ as follows:

$$K_C(x, x') = \sum_{(y,z)\in\mathcal{R}_C(x)} \sum_{(y',z')\in\mathcal{R}_C(x')} K_P(y, y') \cdot K_P(z, z'). \tag{8}$$

The kernel on overlapping parts compares the sets of objects $\mathcal{R}_I(x)$ and $\mathcal{R}_I(x')$ as follows:

$$K_I(x, x') = \sum_{(y,z,w)\in\mathcal{R}_I(x)} \sum_{(y',z',w')\in\mathcal{R}_L(x')} K_P(w, w')\delta(y, y')\delta(z, z') \tag{9}$$

where $\delta(x, y) = 1$ if x and y have the same type and 0 otherwise. The kernel $K_L(x, x')$ on externally connected parts is defined in a similar way:

$$K_L(x, x') = \sum_{(y,z,u,v)\in\mathcal{R}_L(x)} \sum_{(y',z',u',v')\in\mathcal{R}_L(x')} K_P(u, u')K_P(v, v')\delta(y, y')\delta(z, z'). \tag{10}$$

3.3 The general case

Given a set $\mathcal{M}$ of MRs (such as those defined above), the final form of the kernel is

$$K(x, x') = \sum_{M\in\mathcal{M}} K_M(x, x'). \tag{11}$$

Alternatively, a convolution-type form of the kernel can be defined as

$$K(x, x') = \prod_{M\in\mathcal{M}} K_M(x, x'). \tag{12}$$

To equalize the contributions due to different MRs, the kernels K_M can be normalized before combining them with sum or product. It is straightforward to show that K is positive definite, provided kernels defined on attributes are themselves pd, as kernels are closed under both direct sum and tensor product operators [8].

3.4 Remarks

The kernel of Eq. (11) could have been obtained also without the support of logic programming. However, deductive reasoning greatly simplifies the task of recognizing parts and connected parts and at the same time, the declarative style of programming makes it easy and natural to define the features that are implicitly defined by the kernel.

It may be tempting to see a one-to-one mapping between declarative kernels and Haussler's convolution kernels [8]. However the concept of *parts* in [8] is very broad and does not necessarily satisfy mereological assumptions. Eq. 12 can be seen as a convolution kernel but what would be called a *part* in [8] is a mereotopological relation in our case.

4 A guided example: Mutagenesis

Defining and applying declarative kernels involves a three-step process: (1) collect data and background knowledge; (2) interface mereotopology to the available data and knowledge; (3) calculate the kernel on pairs of examples. In this Section, we illustrate the process in a a

```
atm(drug,atom,element,charge,qa)
bond(drug,atom_1,atom_2,bond_kind,qb)
```

b

```
benzene(Drug,RingList) :-
   atoms(Drug,6,AtmList,[c,c,c,c,c,c]),
   ring6(Drug,AtmList,RingList,[7,7,7,7,7,7]).
```

c

```
type(instance).
type(atm).
type(benzene).

obj(X,atm) :-
        atm(Drug,X,_,_,_).
obj(X,benzene) :-
        benzene(Drug,X).
```

d

```
partof(X,X) :-              % P1 axiom
        obj(X,_SomeType).
equalp(X,Y) :-              % P2 axiom
        partof(X,Y), partof(Y,X).
partof(X,Y) :-              % P3 axiom (base)
        has_part(X,Y).
partof(X,Y) :-              % P3 axiom (induction)
        has_part(X,Z), partof(Z,Y).

ppartof(X,Y) :-             % (proper part)
    partof(X,Y), \+ partof(Y,X).

ppartsof(Parts,Y) :-         % MR i)
    setof(X,ppartof(X,Y),Parts).
```

e

```
has_part(B,Drug) :-
        obj(Drug,instance),
        benzene(Drug,B).
```

Figure 2: Code fragments for the guided example (see text).

realistic learning domain. The task (a classical benchmark for relational learners) consists of predicting the mutagenicity of a set of nitro aromatic and heteroaromatic compounds. The first step in this case simply consists of acquiring the atom-bond data and the ring theory developed by Srinivasan *et al.* [15]. Data are described by the two predicates `atm` and `bond` declaring atoms and bonds (Fig. 2a). Background knowledge for this domain consists of a theory describing the main functional groups, such as benzene and nitro (see Fig. 2b for an example). The second step consists of interfacing the available data and knowledge to the kernel. For this purpose, we first need to provide a set of declarations for types, objects, and basic instances of mereotopological relations. Objects are declared using the predicate `obj(X,T)` meaning that `X` is an object of type `T`. For example types include atoms and functional groups (see Fig. 2c). Then we declare basic proper parts via the predicate `has_part(X,Y)` that is true when `Y` is known to be a proper part of `X`. For example if an instance D (a molecule in this case) contains a benzene ring B, then $B \prec_P D$ (Fig. 2d). Note that the use of a predicate called `has_part` (rather than `partof`) is necessary to avoid calling a recursive predicate in the Prolog implementation. The third step is independent of the domain. To calculate the kernel, we first make use of mereotopology to construct the MRs associated with each instance (for example, the code for computing proper parts is shown in Fig. 2e). The resulting sets of ground facts are then passed to a modified version of SVM^{light} [10] for fast kernel calculation.

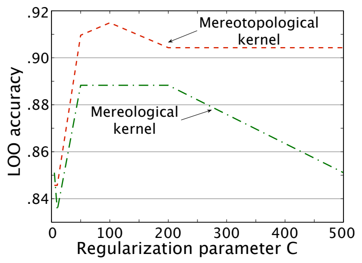
5 Experiments

5.1 Mutagenesis

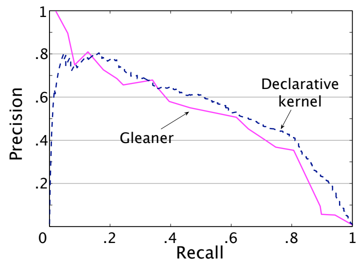
We run a series of 10-fold cross-validation experiments on the regression friendly data set of 188 compounds. First, we applied a mature ILP technique constructing an ensemble of 25 Aleph theories [14]. A 3-fold cross validation in the training set of the first fold was used to select Aleph parameters. We obtained accuracy .88 $\pm$ .07 using atom-bond data and .89 $\pm$ .05 by adding the background ring theory. Next we applied declarative kernels with support vector machines (SVM), obtaining accuracy .90 $\pm$ .07. CPU time was of the order of minutes for the declarative kernel and days for the Aleph ensemble. Finally, we compared the expressive power of ground mereological relations with that of the full mereotopological theory. Fig. 3a reports LOO accuracy for different values of the regularization parameter C, for both mereological and mereotopological kernels, showing the latter achieves both better optimal accuracy and more stable performances.

5.2 Information extraction in biology

In these experiments we apply declarative kernels to the extraction of relational information from free text. Specifically, we focus on multi-slot extraction of binary relations between named entities. Our experiments were carried out on the yeast protein localization data set described in [13] and subsequently used as a testbed for state-of-the-art methods based on ILP [7]. The task consists of learning the rela-



(a)



(b)

Figure 3: (a): LOO accuracy on the regression friendly mutagenesis data set. (b): comparing Gleaner and the declarative kernel on the information extraction task (fold 5).

tion `protein_location` between two named entities representing *candidate* protein names and cell locations. Instances are ordered pairs of noun phrases (NP) extracted from MEDLINE abstracts and with stemmed words. An instance is positive iff the first NP is a protein and the second NP is a location, for example:

```
protein_location("sco1", "the inner mitochondri membran", pos).
protein_location("the ept1 gene product","membran topographi",pos).
protein_location("a reductas activ", "the cell", neg).
protein_location("the ace2 gene", "multipl copi", neg).
```

The data set is a collection of $7,245$ sentences from 871 abstracts, yielding $1,773$ positive and $279,154$ negative instances. The data is enriched by a large body of domain knowledge, including relations about the structure of sentences and abstracts, lexical knowledge, and biological knowledge derived from several specialized vocabularies and ontologies such as MeSH and Gene Ontology. For simplicity, only a fraction of the available knowledge has been used in our experiments. The main data types in this domain are: `instance` (pairs of candidate NP's); `cp_NP` (candidate protein NP); `cl_NP` (candidate location NP); `word_p` (word in a protein NP); `word_l` (word in a location NP). Basic parthood rules in the ontology declare that phrases (`cp_NP` and `cl_NP`) are parts of instances and words are parts of phrases. For this task we used a minimal mereological kernel with no connections and no axiomatic theory to avoid explosion of features due to words appearing both as part of NP's and instances. We compared declarative kernels to state-of-the-art ILP-based system for this domain: Aleph and Gleaner [7]. We used the same setting as in [7], performing a five-fold cross validation, with approximately 250 positive and $120,000$ negative examples in each fold (split at the level of abstracts), and measuring the quality of the predictor by means of the *area under the recall-precision curve* (AURPC). As reported in [7], Aleph attains its best performance (area $.45$) by learning on the order of 10^8 rules, while Gleaner attains similar performance ($.43 \pm .6$) but using several orders of magnitude less rules [6]. We trained five SVMs using the declarative kernel composed with a Gaussian kernel. Gaussian width and the regularization parameter were selected by reserving a validation set inside each fold. The obtained AURPC was $.47 \pm .7$. Fig. 3b compares the recall precision curve reported in [7], which is produced by Gleaner using $1,000,000$ candidate clauses on fold five, with that obtained by the declarative kernel. The result is very encouraging given that only a fraction of the available knowledge has been used. Training took less than three hours on a single 3.00GHz Pentium while Aleph and Gleaner run for several days on a large PC cluster on the same task [6].

5.3 Prediction of mRNA signal structure

We now describe an SVM solution to mRNA classification problem studied by Horwáth et al. [9]. The task consists in predicting the structural class of untranslated regions of mRNA (i.e. portions of mRNA that do not code for proteins). State-of-the-art solutions are in this case based on RIBL, (a k-NN like instance-based algorithm) with an ad-hoc similarity measure. The main data types in this domain are: `instance` (an mRNA signal structure) and mRNA secondary structure types (`hairpin` , `stem` , `bulge3` , `bulge5` , `single`). Every element is part of a structure and adjacent segments are connected. Nucleotides are the attributes of each segment. An SVM with the declarative kernel for this domain achieved 95.4% leave-one-out accuracy on the data set of 66 signal structures (i.e. 3 misclassifications). This result coincides with that reported in [9].

6 Conclusions

Declarative kernels effectively bridge two important paradigms: symbolic and statistical learning. Solving large scale learning problems in domains characterized by complex knowledge (such as the application we presented to information extraction) becomes accessible and relatively easy. In addition, training time compared to state-of-the-art ILP systems can be dramatically improved. Although we have focused on parthood and connection, other relations that do not have a mereotopological nature could possibly incorporated following ideas in this paper. Note also that connection needs not to be restricted to proper parts of instances on which the kernel operates. In facts, the predicate $C(x, y)$ makes sense whenever x and y are objects of the universe and they could be two different instances (e.g. two linked Web pages) or two parts of different instances. This flexibility brings in some potential for supporting in the future frameworks such as collective classification.

References

[1] R. Casati and A. Varzi. *Parts and places: the structures of spatial representation*. MIT Press, Cambridge, MA and London, 1999.

[2] C. M. Cumby and D. Roth. On kernel methods for relational learning. In *Proceedings of the 20th Int. Conf. on Machine Learning (ICML '03)*, 2003.

[3] G. M. Fung, O. L. Mangasarian, and S. Shavlik. Knowledge-based nonlinear kernel classifiers. In *Proc. of COLT/Kernel '03*, volume 2777 of *LNAI*, pages 102–113. Springer, 2003.

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

[5] T. Gärtner. A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, 5(1):49–58, 2003.

[6] M. Goadrich. Personal communication, 2005.

[7] M. Goadrich, L. Oliphant, and J. W. Shavlik. Learning ensembles of first-order clauses for recall-precision curves: A case study in biomedical information extraction. In *Proc. 14th Int. Conf. on Inductive Logic Programming, ILP '04*, pages 98–0115, 2004.

[8] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, 1999.

[9] T. Horváth, S. Wrobel, and U. Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1/2):53–80, 2001.

[10] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT Press, 1998.

[11] S. Kramer, N. Lavrac, and P. Flach. Propositionalization approaches to relational data mining. In *Relational Data Mining*, pages 262–286. Springer-Verlag, 2000.

[12] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.

[13] S. Ray and M. Craven. Representing sentence structure in hidden Markov models for information extraction. In *Proceedings of IJCAI '01*, pages 1273–1279, 2001.

[14] A. Srinivasan. *The Aleph Manual*. Oxford University Computing Laboratory, 2001.

[15] A. Srinivasan, S. Muggleton, M. J. E. Sternberg, and R. D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

[16] L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. MIT Press, 2nd edition, 1994.