Learning with Kernels and Logical Representations

Paolo Frasconi
Machine Learning and Neural Networks Group
Università degli Studi di Firenze, Italy
http://www.dsi.unifi.it/neural

Outline:
A framework for statistical learning in the ILP setting
Kernels on Prolog proof trees
Declarative kernels
kFOIL
Kernels on type extension trees

Statistical logical learning

• Complementary advantages
  • logic can represent domain knowledge in a natural and expressive way
  • inductive logic programming (ILP) can generate theories and explanations
  • probabilities allow us to deal with uncertainty and noise in the data

Learning with logic and probabilities

• An active research area:
  • Probabilistic relational models (Getoor 2001, Pfeffer 2000) and Relational Bayesian networks (Jaeger 1997)
  • Markov logic (Domingos & Richardson 2004, 2006) and Relational Markov networks (Taskar et al. 2002)
  • Stochastic logic programs (Muggleton 1996, 2000; Cussens 1999)
  • Bayesian Logic Programs (Kersting & De Raedt 2001)
  • Relational dependency networks (Neville and Jensen 2004)
  • Many ad-hoc models (e.g. Logical HMMs)
Generative vs. discriminant classifiers

- Generative direction: model the class conditional densities $p(y|x)$, then Bayes’ theorem gives $p(y|x)$
  - SCFG: assign a probability to each production and infer the probability that string $x$ was generated by the grammar
  - Stochastic logic programs generalize SCFGs: assign a probability to each definite clause and infer the probability that a goal is refuted
- Discriminant direction: model $p(y|x)$ directly or even more simply, learn $f(x) = p(Y = 1|x) - 0.5$
  - Logistic regression: $p(Y = 1|x) = \frac{1}{1 + e^{-\theta(x)}}$ where $\phi(x)$ is a feature vector of $x$.

Supervised learning: the classic statistical setting

- Prediction function $f : \mathcal{X} \mapsto \mathcal{Y}$
- Data set: $D = \{(x_i, y_i), i = 1, \ldots, m\}$
- Empirical risk: $l_D[f] = \sum_{(x, y) \in D} V(y_i, f(x_i))$
- Learning: search hypothesis space $\mathcal{H}$
  $$f = \arg \min_{h \in \mathcal{H}} l_D[h] + \mu \cdot \text{complexity}[h]$$

Generative vs. discriminant classifiers

- Input space: $\mathcal{X}$ (an arbitrary set)
- Output space: $\mathcal{Y} = \{-1, 1\}$ (bin. class.), $\mathcal{Y} = \mathbb{R}$ (regression)
- Fixed unknown density $\rho$ on $\mathcal{X} \times \mathcal{Y}$
- The optimal decision would be $\rho(Y = 1|x) > 0.5$
Kernel methods

- Revolutionized machine learning during the last decade
- The “statistical” and the “representational” dimension of learning are naturally decoupled
- Representer theorem
  - e.g. in the case of SVM classification
  \[ f(x) = \sum_{i=1}^{m} \alpha_i y_i K(x, x_i) \]
- Many other algorithms fit in the same framework

Combining kernels and logic

- Develop a flexible approach to kernel design:
  - Background knowledge usually plugged-in via the kernel function
  - Use background knowledge defined by means of logic programs and convert it into a kernel, thus embedding knowledge into statistical learning algorithms in a principled and flexible way.
- Improve accuracy and efficiency of existing systems
- Discover statistically robust features
- Different tasks solvable in the same framework
  - e.g. regression, ranking, etc.

Kernels in relational learning

- Many kernels for discrete data structures
  - Haussler’s seminal work on convolution kernels (1999)
  - Sequences (Lodhi et al., 2002; Jaakkola and Haussler, 1999; Leslie et al., 2002; Cortes et al., 2004)
  - Trees (Collins and Duffy, 2002; Viswanathan and Smola, 2003),
  - Annotated graphs (Gärtner, 2003; Schölkopf and Warmuth, 2003; Kashima et al., 2003; Mahé et al., 2004; Horváth et al., 2004; Swamidass et al. 2005, Menchetti et al., 2005, Costa et al. 2007)

Convolution kernel

(Haussler, 1999)

\[ K_R(x, z) = \sum_{(x_1, \ldots, x_D)}^{\text{part of } x} \sum_{(z_1, \ldots, z_D)}^{\text{part of } z} \prod_{d=1}^{D} K_d(x_d, z_d) \]

Alternatively, can use sum

\[ K_{\text{set}}(x, z) = \sum_{\xi \in x} \sum_{\zeta \in z} K_{\text{member}}(\xi, \zeta). \]

\[ K_{\text{mean}}(x, z) = K_{\text{set}}(x, z) \frac{1}{|x||z|}. \]
Kernels in relational learning

- Many kernels on logical representations
  - Complex individuals defined using higher order logic abstractions (Gärtner et al., 2004)
  - Pure propositionalization (Kramer 2000; Kramer et al., 2001)
  - Description logics + propositionalization (Cumby and Roth, 2002; 2003)
  - Classic ILP learning + propositionalization (Muggleton et al., 2005)

Kernels on Prolog ground terms

- Recursive definition
  - base step: kernel on constants
  - induction:
    - different functors, compare only functors
    - same functors, compare also arguments, recursively
- Can add a type discipline—e.g. to distinguish numerical vs. categorical arguments in compound terms

Example: boiling point of alkanes

Order: put the "backbone" of the molecule on the right spine of the tree

\[
\text{c(h,h,h, c(h,h,h), c(h,h,h), c(c(h,h,h), c(h,h,h), c(h,h,h), c(h,h,h)))}
\]

Straightforward approach based on kernel ridge regression achieves mean error of 3.8 ºC, comparable to highly tuned recursive neural networks

Passerini & Frasconi, IJCAI 2005

declarative kernels

frasconi, passerini, muggleton & lodhi, ILP 05
Declarative kernels

- Declare important relations suitable for defining features
- Convolution kernels based on the part-of relation
- Example:
  
  \[ \text{partof(sentence24, abstract43).} \]
  
  \[ \text{partof(word32, sentence24).} \]
- Additional features can be inferred from an axiomatic theory
  - E.g. \( \text{?partof(word32, abstract43).} \)

mererotopology

- a theory of parts, wholes and connections (Varzi 1996)
- declare instances of main relations explicitly (based on the available background knowledge)
  - e.g. part-of, connected-to
- deduce additional instances of these relations from axioms

<table>
<thead>
<tr>
<th>Axioms (see e.g. Casati &amp; Varzi, 1999)</th>
</tr>
</thead>
</table>

**Extensional mereology:**

\[ x \leq_P x \quad \text{(P1)} \]
\[ x \leq_P y \land y \leq_P x \Rightarrow y = P x \quad \text{(P2)} \]
\[ x \leq_P y \land y \leq_P z \Rightarrow x \leq_P z \quad \text{(P3)} \]
\[ \forall z . (z \leq_P x \Rightarrow \text{Overlap}(z, y)) \Rightarrow x \leq_P y. \quad \text{(P4)} \]

**Mereotopology:**

- Connected(x, x) \quad \text{(C1)}
- Connected(x, y) \Rightarrow Connected(y, x) \quad \text{(C2)}
- \[ x \leq_P y \Rightarrow \forall z . (\text{Connected}(z, x) \Rightarrow \text{Connected}(z, y)) \quad \text{(C3)} \]

Other relations can be inferred, e.g. external connection:

\[ \text{Ext}_{\text{Connected}}(x, y) \text{ iff Connected}(x, y) \land \neg \text{Overlap}(x, y) \]

Example

atoms and functional groups as parts, bonds as basic connections

can infer overlapping parts (e.g. the two rings with common part C3 and C4)

can infer externally connected parts (e.g. nitro and benzene along with terminals C6 and N10)
**Information extraction from text: Protein subcellular localization**

- **Scientific abstracts**
- **Shallow parser for basic sentence segmentation and labeling**
- **Task: predict whether a pair of NP's consists of a protein and a location**

```
protein_location("scol", "the inner mitochondrial membran", pos).
protein_location("the epti gene product", "membrane topogaphi", pos).
protein_location("a reductas activ", "the cell", neg).
protein_location("the ace2 gene", "multipli copi", neg).
```

**Results**

- Only a fraction of available background knowledge
- AURPC = .48 ± .7, compared to .43 ± .6 for Gleaner
- Run in 3 hours on a single CPU (compared to a week on a PC cluster for Gleaner)

**Data set**

- Collected by Ray & Craven (2001) from Pubmed
- 7,245 sentences from 871 abstracts (1,773 positive and 279,154 negative instances)
- Domain knowledge (lexical knowledge, biological knowledge derived from ontologies such as MeSH and Gene Ontology)
- Results compared to state-of-the-art ILP algorithms (Gleaner: Goadrich et al. 2004)

**Interface**

```
\text{has}\text{-}\text{part}(X,\text{Drug}).
\text{obs}(X,\text{Drug},\text{instance}).
\text{benzene}(\text{Drug},X).
```

- Declare types, objects, basic instances of mereotopological relations, such as has\text{-}part
- The sets of proper parts, connected parts, externally connected parts, etc are computed in Prolog and compiled in a domain independent form
- The kernel is then efficiently calculated in C++
Kernels on Prolog proof trees

Passerini, Frasconi & De Raedt, J. Machine Learning Research, 2006

Setting

- Instance space: set of objects (FOL individuals)
- Extensional background theory (facts about specific examples)
- Intensional predicates (relevant to all examples)
- Visitor program
  - goals of the kind visit(x)? are meant to probe examples and extract useful features during the proof

Visitors

- The programs from which example-traces are generated are called visitors
- These programs must be written in addition to the available background knowledge and are meant to probe examples using predicates in the available background knowledge
- The kernel will use the program trace (a proof tree) generated by the visitor program
- The purpose of visitors is thus to construct useful features during their execution.
Program traces

- A visitor program trace is obtained by recording proofs of visitor goals called on that instance.
- Since we start from Prolog-based representations, a natural option would be the use of SLD-trees, whose paths correspond to execution sequences of the Prolog interpreter.
- However, SLD-trees are too complex and unstructured.
- In addition, failed paths may lead to an explosion of redundant and irrelevant features.
- For these reasons we prefer to resort to proof trees.

Proof trees

- In general not ground but we will define a kernel on ground trees.
- A tree can always be grounded either by skolemization.
- A sufficient condition for ground proofs is that all clauses are range-restricted (i.e. that all variables that appear in the head of a clause also appear in its body).
- Ground proofs can be also obtained by making assumptions about the mode of head variables not occurring in the body, so that these variables will be instantiated when proving the goal.

Trace

Given visitors \( V_1, V_2, \ldots, V_N \):
- \( T_{j,x} \): proof tree for the \( j \)-th proof of goal \( V_i(x) \)
- Proof forest:
  \[ T_{j,x} = \{ T_{j1,x}, \ldots, T_{jN,x} \} \]
- Trace of \( x \):
  \[ T_x = [ T_{1,x}, \ldots, T_{N,x} ] \]

the trace is a tuple of proof forests
Pruning

- Details on how some predicates are proved may be not relevant
- E.g. we might believe that spotting a benzene ring is already a good feature, so don’t need a trace for benzene/2
- Use leaf/1 to declare that the proof tree must be pruned
- This will reduce the complexity of the feature space associated with the kernel

\[
\text{visit}_{-}\text{benzene}(X):=
\begin{align*}
\text{benzene}(X,\text{Atoms}), \\
\text{atoms}(X,\text{Atoms}).
\end{align*}
\]

\[
\text{atoms}(X,[]).
\]

\[
\text{atoms}(X,[\text{H}_2]):=
\begin{align*}
\text{atom}(X,\text{H},[],\text{..}), \\
\text{atoms}(X,\text{F}).
\end{align*}
\]

\[
\text{leaf}_{-}\text{benzene}(\_,\_)).
\]

Proof trees as Prolog ground terms

- Motivations:
  - we can exploit type information on constants and functors so that different sub-kernels can be applied to different object types
  - can be more selective on which parts of the proof should be used in the comparison:
    - traditional tree kernels — e.g. as in Collins and Duffy (2002), or Viewanathan and Smola (2003) — would typically compare all pairs of subtrees between two proofs
    - but we may want to compare certain parts of two proofs only when reached by following similar inference steps (a distinction that would be difficult to implement with traditional tree kernels)

Kernels on traces

- Top-down approach. Begin by decomposing traces into parts associated with different visitors:

\[
K(T_x, T_z) = \sum_{\ell=1}^{N} K_{\ell}(T_{\ell,x}, T_{\ell,z}).
\]

where \( K_{\ell} \) is a kernel on proof forests

- Apply the set kernel:

\[
K_{\ell}(T_{\ell,x}, T_{\ell,z}) = \sum_{p=1}^{n_x} \sum_{q=1}^{n_z} K_{\text{tree}}(T_{\ell,p,x}, T_{\ell,q,z}).
\]

so the problem boils down to defining a kernel \( K_{\text{tree}} \) over individual proof trees.

Example
n-of-m problems

background knowledge

\[
\text{substr}(A,B,[A,B|T]),
\quad \text{substr}(A,B,[H|T]):=\text{substr}(A,B,T).
\]

\[
\text{comp}(A,B):= A \neq B,
\quad \text{comp}(A,B):= A \preceq B.
\]

setting

train:
300 random sequences
length < 6

test:
1455 random sequences
6 \leq \text{length} \leq 100

visitor

\[
\text{visit}(X):=\text{string}(X,S),\text{substr}(A,B,S),\text{comp}(A,B).
\]

results

trace kernel: [1, 1, 1, 1, 528, 0, 94, 633, True]

prolog, tilde:
no results, except after adding explicit counting predicates

Regression tasks

- Regression is usually harder than classification for ILP systems
- We studied two prototypical regression problems:
  - QSAR (mutagenesis domain)
  - biodegradability
- Kernel ridge-regression (KRR) using trace-kernels

m-of-n problems

- the model must learn to count and make a decision according to the count
- usually very hard for ILP methods
- artificial problem:
  - decide if a sequence is "mostly sorted" (> 50% consecutive pairs ordered)
  - sequences of integers in 0..9, standard \leq order relation
Mutagenesis

regression friendly data set, 188 compounds as in Srinivasan et al. (1996)

Kernel ridge regression (KRR) significantly better than most ILP and propositional regression approaches

---

<table>
<thead>
<tr>
<th>System</th>
<th>correlation coefficient</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRR</td>
<td>0.898 (0.002)</td>
<td></td>
</tr>
<tr>
<td>S-CART</td>
<td>0.830 (0.020)</td>
<td></td>
</tr>
<tr>
<td>P + S-CART</td>
<td>0.834 (0.010)</td>
<td></td>
</tr>
<tr>
<td>P + M₅</td>
<td>0.893 (0.001)</td>
<td></td>
</tr>
<tr>
<td>P + SP + S-CART</td>
<td>0.767 (0.038)</td>
<td></td>
</tr>
<tr>
<td>P + SP + M₅</td>
<td>0.835 (0.012)</td>
<td></td>
</tr>
</tbody>
</table>

★ S-CART (Kramer, 1999): upgrades CART to FOL
★ P: propositionalization counting occurrences of functional groups
★ SP: supervised stochastic propositionalization algorithm by Kramer (1999)

---

Protein fold classification

- Problem initially formulated and studied by Turcotte et al. (2001)
- Task: classifying proteins of known structure (i.e. proteins experimentally solved and deposited in the Protein Data Bank) into SCOP folds, given their high-level logical descriptions about secondary structure and amino acid sequence
- SCOP (http://scop.mrc-lmb.cam.ac.uk/scop/) is a manually curated database of proteins hierarchically organized according to their structural properties

---

The structure of SCOP

- Top level of the SCOP hierarchy: proteins are grouped into classes
- Four main classes are all-α, all-β, α/β, and α + β
- Each class is then divided into folds (similar secondary structures and three-dimensional arrangements)
- The hierarchy is further refined by dividing folds into superfamilies, then families and eventually protein domains
- Here we are only interested in the two top levels of the hierarchy: classify proteins into the correct fold

---

The data set

Entries from the five most populated folds in each of the four main SCOP classes: 20 binary classification problems
Positive examples: target fold
Negative examples: same class but different folds
20 small data sets (30–160 examples per fold, totaling 1143 examples).
Background knowledge

- Global predicates
  - Domain length, number of helices and strands (suitable also for propositional learners)
- Relational (connection) predicates
  - e.g. adjacency between secondary structure elements
- Local predicates
  - e.g. length of loops, hydrophobicity, presence of proline, etc

Visitors

- There are global, connection and local predicates in the BK
- Hence we have global (a), connection (b) and local (c) visitors

Visitor 3

```
visit_global(X):-
    normstr(X,N),
    normstr(X,NumAlpha),
    normstr(X,NumBeta),
    has_pred(X).
```

Visitor 2

```
visit_adjacent(X) :-
    adjacent(X, A, B, Pred_Type),
    normstr(A, NumAlpha),
    unit_features(X).
```

Visitor 1

```
visit_unit(X) :-
    unit_features(X),
    leaf(normstr(X,NumAlpha)).
```

Multiclass protein fold classification

- Same data set as before but in the multiclass setting used by Chen et al. (2006)
- For each of the main SCOP classes (all-α, all-β, α/β, and α + β) we have a 5-class classification problem
- Plain one-vs-all multiclass SVM with the trace kernel
- Experimental comparison wrt stochastic logic programs (Chen et al. 2006) that improved significantly accuracy and understandability over progol

---

Results (10-fold cross-validation)

<table>
<thead>
<tr>
<th>Fold</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>93.8</td>
<td>92.3</td>
<td>93.0</td>
</tr>
<tr>
<td>2</td>
<td>94.0</td>
<td>93.7</td>
<td>93.9</td>
</tr>
<tr>
<td>3</td>
<td>94.1</td>
<td>93.8</td>
<td>93.9</td>
</tr>
<tr>
<td>4</td>
<td>94.2</td>
<td>93.9</td>
<td>94.0</td>
</tr>
<tr>
<td>5</td>
<td>94.3</td>
<td>94.0</td>
<td>94.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fold</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>94.4</td>
<td>94.1</td>
<td>94.2</td>
</tr>
<tr>
<td>7</td>
<td>94.5</td>
<td>94.2</td>
<td>94.3</td>
</tr>
<tr>
<td>8</td>
<td>94.6</td>
<td>94.3</td>
<td>94.4</td>
</tr>
<tr>
<td>9</td>
<td>94.7</td>
<td>94.4</td>
<td>94.5</td>
</tr>
<tr>
<td>10</td>
<td>94.8</td>
<td>94.5</td>
<td>94.6</td>
</tr>
</tbody>
</table>

---

**Average**

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>94.5</td>
<td>94.3</td>
<td>94.4</td>
</tr>
</tbody>
</table>

95% confidence intervals
Multiclass results

- 5-fold cross validation as in Chen et al. 2006 (same splits)

- Both SLP and SVM with proof tree kernels improved significantly over non-probabilistic ILP

- SVM slightly better than SLP (not significant)

<table>
<thead>
<tr>
<th>fold class</th>
<th>SVM</th>
<th>SLP</th>
<th>Progol</th>
</tr>
</thead>
<tbody>
<tr>
<td>all-α</td>
<td>80.5±4.5 (62/77)</td>
<td>76.6±4.8 (59/77)</td>
<td>71.4±5.2 (55/77)</td>
</tr>
<tr>
<td>all-β</td>
<td>87.1±3.1 (101/116)</td>
<td>81.0±3.6 (94/116)</td>
<td>69.8±4.3 (81/116)</td>
</tr>
<tr>
<td>α/β</td>
<td>61.7±4.5 (71/115)</td>
<td>51.3±4.7 (59/115)</td>
<td>44.4±4.6 (51/115)</td>
</tr>
<tr>
<td>α + β</td>
<td>60.3±5.7 (44/73)</td>
<td>82.2±4.5 (60/73)</td>
<td>80.8±4.6 (59/73)</td>
</tr>
<tr>
<td>overall</td>
<td>73.0±2.3 (278/381)</td>
<td>71.4±2.3 (272/381)</td>
<td>64.6±2.5 (246/381)</td>
</tr>
</tbody>
</table>

Discussion

- Main ingredients
  - background theory—entirely similar to ILP systems
  - declarative approach for defining features associated with parts and connections
  - visitor—a new form of bias: declarative bias in ILP is purely syntactic while the visitor can be procedural;
  - statistical learning—noise robustness difficult to achieve in ILP

- Pros
  - Accuracy (due to noise robustness)
  - Expressivity (due to visitor bias)
  - Efficiency (SVM and ridge regression are fast given the Gram matrix)
  - Regression can be solved naturally

- Cons
  - Predictions are opaque and no feature discovery

kFOIL: Learning Simple Relational Kernels
Motivations

- So far the kernel is fixed in advance (before learning)
- However, similarity measure is problem-dependent!
- kFOIL: learn the kernel from data
  - kernel defined by a small set of clauses
  - clauses learned from data, ILP style
- Bring, to some degree, interpretability to kernel machines

Relational Kernels

- ILP
  - Expressive, Accurate
  - Interpretable, General-purpose

Relational Data and Clausal Theories

Clauses match on structural features of e.g. molecules
Set of clauses is called theory or hypothesis

Logical theories

The theory $H = \{c_1, \ldots, c_k\}$ can be seen as a set of features

$$
\phi_H(x) = \begin{pmatrix}
\pi_1 \\
\pi_2 \\
\vdots \\
\pi_k
\end{pmatrix}
\pi_i = \begin{cases}
1 & \text{if } c_i \text{ matches } x \\
0 & \text{otherwise}
\end{cases}
$$
Example

\[ \phi_H(x_1) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \]

\[ \phi_H(x_2) = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \]

Example

\[ K(x_1, x_2, H) = \langle \phi_H(x_1), \phi_H(x_2) \rangle = 1 \]

Clause-based Kernels

- Use theory to measure similarity
  - Similarity is high if clauses behave similarly on examples
  - Define the kernel between two examples as the number of clauses in the theory that fire simultaneously

\[ K(x, x', H) = \langle \phi_H(x), \phi_H(x') \rangle \]

- cf. also (Muggleton 2005)

Kernel composition

- Polynomial kernel
  \[ K(x, x', H) = \langle \phi_H(x), \phi_H(x') \rangle^d \]
  interpreted as the conjunction of up to \( d \) clauses

- Gaussian kernel
  \[ K(x, x', H) = \exp \left( - \frac{\| \phi_H(x) - \phi_H(x') \|^2}{2\sigma} \right) \]
  interpreted as the number of clauses firing in one and only one example
"Classic" ILP vs. kFOIL

- **Given:**
  - Set of labeled examples $E$
  - Background knowledge $B$
  - Language of clauses $L$ (language bias)

- **Find:**
  - A hypothesis $H \subseteq L$ that covers all the positive examples and none of the negative examples
  - A hypothesis $H \subseteq L$ that maximizes score($E, B, H$), defined as the performance of a kernel machine based on $K$

---

More on the score

- **Margin:**
  
  $$
  f(x) = \sum_{i=1}^{m} \alpha_i y_i K(x, x_i, H) + b
  $$

  - Binary classification

  $$
  f(x) = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) y_i K(x, x_i, H) + b
  $$

  - Regression

- A simplest measure is classification accuracy or RMSE

---

Learning $H$

- **Traditional static propositionalization** approaches
  - ILP + SVM
    1. Run ILP system to generate small set of clauses
    2. Train SVM based on these features
  - Pattern Mining + SVM
    1. Generate large set of clauses, e.g., all clauses that are frequent in the training data
    2. Train SVM based on these features

- In both cases, steps (1) and (2) are independent, the best ILP theory does not necessarily maximize the score

---

Learning $H$

- **Dynamic propositionalization** (Landwehr et al. 2006)
  - Steps (1) and (2) are integrated
  - The ILP algorithm is guided by performance of the kernel machine

- Search in kFOIL:
  - FOIL style greedy general-to-specific search for clauses employing refinement operator

  $$
  \text{refine}(c) = \{c' | c' \text{ is a minimal specialization of } c \text{ in } L\}$$
Experiments

- kFOIL + polynomial kernel of degree 2
- Comparison against:
  - Aleph: state-of-the-art ILP system
  - c-ARMR (De Raedt & Ramon 2004) + SVM: Static propositionalization using all frequent clauses
  - nFOIL (Landwehr et al. 2005): dynamic propositionalization system integrating FOIL and Naïve Bayes
- Several ILP benchmark datasets

Results (binary classification)

- kFOIL significantly better (95% confidence)

Results (regression)

- kFOIL significantly better (95% confidence)

Ongoing extensions

- Computational efficiency
  - Incremental update of portions of the Gram matrix affected by each refinement
- Multi-task learning
  - Learn a kernel that solves several tasks simultaneously
Motivations

• Comparing two methods for relational data:
  • ILP
    • General-purpose
    • Simple and interpretable models
  • Kernels on relational data
    • Specifically designed for the problem at hand
    • The resulting machine is opaque
  • Can we combine the advantages of the two approaches?