Introduction to PILP

Bernard ESPINASSE
Aix-Marseille Université (AMU), France.

Rinaldo LIMA
Federal Rural University of Pernambuco (UFRPE), PE, Brazil.

20 mars 2017

• Statistical Relational Learning (SRL)
• Probabilistic Inductive Logic Programming (PILP)
• KLog: a PILP system
• KLogNLP: KLog for Natural Language Processing

Outline

1. Statistical Relational Learning (SRL)
   • From Logical Relational Learning (LRL) to Statistical Relational Learning (SRL)
   • SRL Approaches & Models
   • SRL Structure & Parameters Learning
   • Comparisons of SRL Models

2. Probabilistic Inductive Logic Programming (PILP)
   • From ILP to PILP
   • PILP Definition, Interests & Programming Settings
   • PLIP in Parameter & Structure Learning
   • Some PLIP Systems
   • Strengths and Weakness of PILP

3. KLog: a PILP system
   • KLog definition, workflow & specificities
   • KLog: an example of application

4. KLogNLP: KLog for Natural Language Processing
   • KLogNLP Definition & Workflow
   • KLogNLP Declarative Model
   • KLogNLP: an Example of Application

References

Books, articles and reports:

Course/tutorials:
• Course of Krishnaprasad Thirunarayan (T.K.Prasad), Wright University, Dayton, Ohio (USA).
• Cours de C. Vrain, Université d’Orléans (France).
• Slides of L. de Raedt, ICLP 2015.
Logical & Relational Learning: Principle

Goal: to find a hypothesis \( h \), i.e., a logic program, from a set of positive and negative examples.

- Given:
  - a set of training examples \( T \) expressed in a language chosen for representing the examples \( L_E \),
  - a background knowledge \( B \),
  - a hypothesis language \( L_H \) that specifies the clauses that are allowed in the hypotheses,
  - a relation \( \text{covers}(e, H, B) \) which determines the classification of an example \( e \) with respect to \( H \) and \( B \).

- Find a hypothesis \( h \in H \) that:
  - covers all positive training examples and
  - none of the negative ones with respect to background theory \( B \).

---

SRL (Statistical Relational Learning)

- **SRL** is a sub-discipline of AI and Machine Learning concerned with domain models that exhibit both:
  - Uncertainty (which can be dealt with using statistical methods)
  - Complex and relational structure

- Significant contributions in the late 1990s

- Knowledge representation formalisms in SRL use:
  - A subset of first-order logic to describe relational properties of a domain in a general manner (universal quantification)
  - Probabilistic graphical models (as Bayesian networks or Markov networks) to model the uncertainty;
  - Some also are build upon **Inductive Logic Programming (ILP)**: Probabilistic ILP – PILP

---

Logical & Relational Learning: learning settings

- Specific learning setting is determined by \( L_E \) language together with the \( \text{covers} \) relation [De Raedt, 1997]

- Most popular learning settings are:
  - **Learning from entailment** [Plotkin 1970]: the examples are definite clauses:
    - An hypothesis \( h \) covers an example \( e \) with respect to the background knowledge \( B \) if and only if \( B \cup H \models e \)
    - An example can consist of just a single fact.
  - **Learning from interpretations** [De Raedt and Dzeroski, 1994]: the examples are Herbrand interpretations:
    - An hypothesis \( h \) covers an example \( e \) with respect to the background knowledge \( B \) if and only if \( e \) is a model of \( B \cup H \)
    - All facts that hold in the example are known, more information is available to the learner.

---

SRL approaches

- **Combining Logic and Probability in Machine Learning**:

  ![Diagram of SRL approaches](image)

  - **Approach 1**: Logic → Add Probabilities → Statistical Relational Learning (SRL)
  - **Approach 2**: Probabilities → Add Relations → Statistical Relational Learning (SRL)
Statistical Relational Learning Main Models

Popular SRL models include:

**Approach 1: Probability + Logic**
- Probabilistic Relational Models (PRM) [Friedman et al. 1999]
- Bayesian Logic Programs (BLP) [Kersting & De Raedt 2007] can be seen as the extensions of Bayesian Networks and Relational Markov Networks [Taskar et al. 2007]
- Markov Logic Networks (MLN) [Richardson & Domingos 2006] are the extensions of Markov Networks.
- Relational Dependencies Network (RDN) [Neville and Jensen, 2007] extend Dependency Networks (DN) [Heckerman et al., 2001] to the relational setting.

**Approach 2: Logic + Probability**
- PILP (Probabilistic Inductive Logic Programming) [Raedt and Kersting 2004] an extension of ILP which can deal with uncertainty, BK and examples can be annotated with probabilities.

### Comparison of SRL Models

According various dimensions of importance in SRL [Khosravi & Bina 2010]:
- Probabilistic Relational Models (PRM),
- Markov Logic Networks (MLN),
- Relational Dependency Networks (RDN),
- Bayes Logic Networks (BLN).

<table>
<thead>
<tr>
<th>Class level model</th>
<th>PRM</th>
<th>MLN</th>
<th>RDN</th>
<th>BLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct GM</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Logical clauses</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Bidirected GM</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Bipartite Directed GM</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter estimation</th>
<th>PRM</th>
<th>MLN</th>
<th>RDN</th>
<th>BLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML to fill CPTs</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>ML to learn weights</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>CR learners to CPTs</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>ML to fill CPTs</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Structure learning</th>
<th>PRM</th>
<th>MLN</th>
<th>RDN</th>
<th>BLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score based learning</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>ILP methods</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Use CR learners</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>ILP methods</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inference graph</th>
<th>PRM</th>
<th>MLN</th>
<th>RDN</th>
<th>BLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bayesian networks</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Markov model</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Undirected model</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Bayesian network</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inference method</th>
<th>PRM</th>
<th>MLN</th>
<th>RDN</th>
<th>BLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belief propagation</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Pseudo-likelihood</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Pseudo-likelihood</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Stand BN inference</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X-many relationships</th>
<th>PRM</th>
<th>MLN</th>
<th>RDN</th>
<th>BLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require aggregation</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>No requirements</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>No requirements</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Require combination rules</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

- MLN, PRM and BLN learn the structure of their models through techniques similar to those of ILP,
- RDNs use relational classifiers developed in the ILP community as a subroutine to its model learning [Braz et al., 2008].

### SRL: Structure & Parameters Learning

<table>
<thead>
<tr>
<th>Example representation</th>
<th>Learning model representation</th>
<th>Structure learning</th>
<th>Parameter learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table representations</td>
<td>Probability</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>(propositional)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Markov Logic Network (MLN)</th>
<th>Symbolic</th>
<th>Symbolic Rules</th>
<th>NO* (limitations)</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational Dependency Networks (RDN)</td>
<td>Symbolic &amp; Integrates BK</td>
<td>Symbolic Rules</td>
<td>YES* (The type of the structure is fixed)</td>
<td>YES</td>
</tr>
<tr>
<td>PILP (KLog /KLogNLP)</td>
<td>Symbolic</td>
<td>Probability</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>nFOIL, QuickFOIL, ProbFOIL (De Raedt Group)</td>
<td>Symbolic</td>
<td>Symbolic Rules</td>
<td>FOIL: YES (just propositional symbolic rules. ProbFOIL is relational.)</td>
<td>YES</td>
</tr>
</tbody>
</table>

### 2. PILP (Probabilistic Inductive Logic Programming)

- From ILP to PILP
- PILP Definition, Interests & Programming Settings
- PILP in Parameter & Structure Learning
- Some PILP Systems
- Strengths and Weakness of PILP
The “Rock-paper-scissors” game
(from Joana Corte-Real et al.)

<table>
<thead>
<tr>
<th>rock</th>
<th>paper</th>
<th>scissors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Rules**
- beats (Round, PlayerA, PlayerB) :-
  - plays (Round, PlayerA, rock),
  - plays (Round, PlayerB, scissors)

**Facts (Background Knowledge - BK)**
- plays(1, ines, rock)
- plays(1, joana, scissors)

**Examples**
- beats(1, ines, joana)

- **Rules**: rule(s) to win the game
- **Facts (BK)**: who and what each player plays at each round
- **Example**: who win at each round

---

**Inductive Logic Programming (ILP)**

- **ILP is a technique related to Logical and Relational Learning**:
  - At the intersect of **Machine Learning** & **Logic Programming** domains

- **Which learns logic rules from examples and background knowledge (BK)**
  
  Ex: learn the rule for grand parents, given background knowledge of parents and examples of grandparents

- **Induces rules which explain examples and BK**
- **based on Logic Programming (Prolog)**

- **ILP can be used for**:
  - **Classification** and **Prediction**
  - to **interface with experts** of other areas of knowledge

---

**ILP illustration in “Rock-paper-scissors” game**

- **Inductive Reasoning**: can learn a rule from examples and a set of facts which describe the example (or Background Knowledge - BK)

<table>
<thead>
<tr>
<th>Facts (BK)</th>
<th>plays(1, ines, rock)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>plays(1, joana, scissors)</td>
</tr>
</tbody>
</table>

+ **Examples**
- beats(1, ines, joana)

= **Rules**
- beats (Round, PlayerA, PlayerB) :-
  - plays (Round, PlayerA, rock),
  - plays (Round, PlayerB, scissors)

⇒ **ILP (Inductive Logic Programming)**

---

**Formal definition of ILP**

- **Given**:
  - a logic program $B$ representing **Background Knowledge (BK)**
  - a set of **positive examples** $E^+$
  - a set of **negative examples** $E^-$

- **Find hypothesis** $H$ such that:
  1. $B \cup H \models e$ for every $e \in E^+$
  2. $B \cup H \models f$ for every $f \in E^-$
  3. $B \cup H$ is consistent
  4. Assume that $B \not\models e$ for some $e \in E^+$
From ILP to PILP

**ILP + Probabilities = PILP**

*Logical & Relational Learning (LRL) ➔ Statistical Relational Learning (SRL)*

---

**PLIP illustration in “Rock-paper-scissors » game**
(from Joana Corte-Real et al. 2015)

<table>
<thead>
<tr>
<th>PLIP (%)</th>
<th>PILP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facts (BK)</td>
<td>0.7:plays(ines,rock) 0.3:plays(ines,scissors) 0.5:plays(joana,rock) 0.5:plays(joana,scissors).</td>
</tr>
<tr>
<td>Examples</td>
<td>beats(1,ines,joana) 0.35:beats(ines,joana)</td>
</tr>
<tr>
<td>Rules</td>
<td>beats(Round,PlayerA,PlayerB) :- plays(Round,PlayerA,rock), plays(Round,PlayerB,scissors).</td>
</tr>
</tbody>
</table>

---

**PILP**

- **PILP definition**
  - Probabilistic Inductive Logic Programming (PILP) is an ILP extension combining [De Readt & Kersting, 2008]:
    - the expressiveness of inductive logic learning with probabilistic reasoning
  - **PILP specifications / ILP:**
    - BK and examples can be annotated with probabilities
    - **Semantic of probabilities in PILP:**
      - Statistics: statistical data such as frequencies, marginal distributions, ...
      - Confidence: or the degree of belief in a statement (can be used to interface with human experts)
      - Adaptation: from a different type of data such as a numerical attribute.

---

**PLIP Programming Settings**

PILP we have 2 changes in comparison of PLI:

1. **clauses are annotated with probability values**, 
2. **covers relation becomes a probabilistic cover relation**

- a probabilistic covers relation $P(e | H, B)$:
  - takes as arguments: an example $e$, a hypothesis $H$ and possibly the Background Knowledge (theory) $B$, 
  - returns a probability value between 0 and 1.
- **Task of PILP**: find the hypothesis $H^*$ that maximizes the probability $P$:

$$P(E | H^*, B) = \prod_{e \in E} P(e | H^*, B)$$

where $E$ denotes the set of examples.
Formal definition of PILP (1)

PILP can be defined as follows [DeRealdt & Kersting, 2008]:

Given

- a set of examples $E$
- a probabilistic covers relation $P(e \mid H, B)$
- a probabilistic logic programming representation language
- possibly a Background Knowledge (Theory) $B$

Find the hypothesis $H^* = \text{argmax}_H P(E \mid H, B)$

(Where $\text{argmax}$ are the points of the domain of $P$ at which the $P$ values are maximized)

With $H = (L, \lambda)$: a logic program $L$ annotated with probabilistic parameters $\lambda$

2 subtasks distinguished:

- **Parameter learning**, where it is assumed that the underlying logic program $L$ is fixed, and the learning task consist of estimating the parameters $\lambda$ that maximize the likelihood.
- **Structure learning**, where both $L$ and $\lambda$ have to be learned from the data.

Probabilistic ILP (PLIP) interests

- **Main PILP interests:**
  - **Better models**: many fields of knowledge produce data with an inherent uncertainty
  - **Dataset size**: using probabilities to describe data can greatly reduce the dataset size and useful information can still be extracted
  - **Unknown values**: in cases where the full data is not known, information can still be used efficiently in the computation of a rule, for instance, by adding values from the literature
  - **Privacy**: compressing data this way can be used to protect private sensitive data
  - ...

Formal definition of PILP (2)

PILP can be defined as follows [Riguzzi et al., 2014]:

- Given:
  - Background Knowledge as a probabilistic logic program $B$
  - a set of positive and negative examples $E^+$ and $E^-$
  - a language bias $L$

Find:

- a probabilistic logic program $P$ such that:
  - the probability of positive examples $E^+$ according to $P \cup B$ is maximized and
  - the probability of negative examples $E^-$ is minimized.

This problem has 2 variants:

- **Parameter learning**: given the structure (the rules) of $P$, we just want to infer the parameters of $P$
- **Structure learning**: infer both the structure and the parameters of $P$.

PILP in Parameter Learning

Parameter learning: given the structure (the rules) of $P$, we just want to infer the parameters of $P$

PILP systems:

- use the EM (Expectation Maximization) algorithm to estimate the probability of models containing random variables that are not observed in the data.
- the EM algorithm consists of a cycle in which the steps of Expectation and Maximization are repeatedly performed:
  - in **Expectation step**, the distribution of the hidden variables is computed according to the current values of the parameters,
  - in the **Maximization step**, the new values of the parameters are computed, using in general knowledge compilation for computing the distribution of the hidden variables
- Gradient descent methods compute the gradient of the target function and iteratively modify the parameters moving in the direction of the gradient.
- Example: LeProbLog (Gutmann et al., 2008) that uses a dynamic programming algorithm for computing the gradient exploiting BDDs.
PILP in Structure Learning

Structure learning: infer both the structure and the parameters of the statistical model.

PILP systems:
- some systems [Koller and Pfeffer, 1997] learn the structure of first-order rules with associated probabilistic uncertainty parameters. Their approach involves generating the underlying graphical model and then EM is applied on the graphical model.
- [De Raedt et al. 2008] presented an algorithm for performing theory compression on ProbLog programs. Theory compression means removing as many clauses as possible from the theory in order to maximize the probability. No new clause can be added to the theory.
- ProbFOIL (De Raedt and Thon, 2010) combines the rule learner FOIL (Quinlan and Cameron-Jones, 1993) with ProbLog. Logical rules are learned from probabilistic data in the sense that both the examples themselves and their classifications can be probabilistic. The set of rules has to allow predicting the probability of the examples from their description. In this setting, the parameters (the probability values) are fixed and the structure has to be learned.

Some PILP Systems

Some PILP systems:
- kLog [Frasconi et al., 2014] is a language for logical and relational learning with kernels. It is embedded in Prolog, and builds upon and links together concepts from database theory, logic programming and learning from interpretations.
- ProbLog [Costa et al., 2011] is a probabilistic extension of Prolog, implemented in YAP Prolog, permitting to work with probabilistic facts and background knowledge.
- ProbFOIL [Raedt and Thon, 2010] uses ProbLog to evaluate probabilities of examples given a learned hypothesis P(H,u B e), based on probabilistic facts and non-probabilistic rules.
- Skill [Corte-Real et al., 2015] is a Stochastic Inductive Logic Learner which can combine the rule learning capability of classic ILP with uncertain knowledge by generating FOL rules based on a richer and more expressive representation of the data than traditional ILP supports.

SRL and Natural Language Processing (NLP)

Statistical relational learning (SRL) of natural language is a research area at the intersection of computational natural language learning and statistical relational learning:

- uses a language which is easy to interpret for experts from other areas of knowledge
- very concise classifiers
- great representative capacity: can represent relations and uncertainty

Strengths and Weakness of PILP

Strengths of PILP:
- presents efficiency issues
- data may not be easy to adapt
3. KLog: a PILP system

- KLog definition, workflow & specificities
- KLog: an example of application


---

kLog definition

- kLog is a logical and relational language for kernel-based learning framework
- kLog permits to specify logical and relational learning problems at a high level in a declarative way
- kLog is based on simple but powerful concepts:
  1. learning from interpretations,
  2. entity/relationship data modeling,
  3. logic programming
  4. deductive databases (Prolog and Datalog),
  5. graph kernels.

---

kLog Workflow

kLog workflow [Verbeke 2014]:

- Feature extraction
- Declarative feature construction
- Graphicalization
- Feature generation
- Graph kernel (NSPK)
- Kernel matrix: feature vectors
- Statistical learner

---

kLog Specificities (1)

- Unlike other Statistical Relational Learning models, kLog does not represent a probability distribution directly
- It is rather a kernel-based approach to learning that employs features derived from a grounded entity/relationship diagram.
- These features are derived using a novel technique called graphicalization:
  1. Relational representations are transformed into graph based representations,
  2. Then graph kernels are employed for defining feature spaces.
kLog Specificities (2)

- **kLog can use:**
  - numerical and symbolic data,
  - background knowledge in the form of Prolog or Datalog programs (as in ILP systems) and
  - several statistical procedures can be used to fit the model parameters.
- The kLog framework can be applied to **tackle the same range of tasks that has made SRL so popular**, including:
  - classification,
  - regression,
  - multitask learning, and
  - collective classification.
  - ...

Use of kLog step by step

Use of kLog step by step

kLog example: defining the game (1)
(source: [Verbeke 2014])

**Defining the game**
Consider a bag filled with blocks, which can be either cubes or spheres, each with a particular color and weight. Each block also has a identifier, which uniquely characterizes the object.

The game consists of drawing a random sequence of 5 objects from the bag. The game can only be won if 2 spheres with the same color are drawn without drawing 2 cubes with the same color.

In any other case the game is lost.

This logic program enumerates all spheres and cubes in the bag, and states that 2 cubes have the same color:

```
sphere(s1,r,2) sphere(s2,g,2) sphere(s3,g,3) sphere(s4,r,1) cube(c1,g,1). cube(c2,r,2). cube(c3,g,4). cube(c4,r,3) cub_eq_col(CubeX,CubeY) :- cube(CubeX,Color,_), cube(CubeY,Color,_) :- cube(c2,r,2). cube(c3,g,4).
```

kLog example: E/R modelling – **graphicalization process** (1)

- As kLog employs a learning from interpretations setting, each interpretation is a set of tuples that are true in the example.
- In kLog each interpretation can be seen as an instance of a (small) relational database, and an E/R model gives an abstract representation of the interpretations, the E/R diagram for the artificial game:
- kLog assumes a **closed world**, which means that atoms that are not known to be true, are assumed to be false.

Note that one could also model spheres and cubes as a single object entity with an additional attribute that represents the shape. Similarly, different relations were used to model the sequence in which the entities are drawn from the bag.
**KLog example: E/R modelling – graphicatization process (2)**

Example interpretation for one particular instance of the game is:

- sphere(s1,r,2)
- cube(c1,g,1)
- sphere(s2,g,2)
- sphere(s3,g,3)
- cube(c2,r,2)

<table>
<thead>
<tr>
<th>sphere(s1,r,2)</th>
<th>next_SC(s1,c1)</th>
<th>diff_shape_eq_weight(s1,c2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cube(c1,g,1)</td>
<td>next_SC(c1,c2)</td>
<td>sph_eq_col(s2,s3)</td>
</tr>
<tr>
<td>sphere(s2,g,2)</td>
<td>next_SC(s2,s3)</td>
<td>diff_shape_eq_weight(s2,c2)</td>
</tr>
<tr>
<td>sphere(s3,g,3)</td>
<td>next_SC(s3,c2)</td>
<td></td>
</tr>
</tbody>
</table>

- sphere(s1,r,2) represents a red sphere with identifier s1 and a weight of 2 kg
- next relations represent the sequence in which the entities are drawn from the bag during the game (a next_s relation is present between spheres if they are drawn consecutively from the bag).
- sph_eq_col relation indicates that two spheres have the same color
- diff_shape_eq_weight relation indicates that a sphere and a cube of different shape, but have an equal weight.

The target relation is the one that needs to be predicted by the classifier, and represents if a game is won or lost.

**KLog example: declarative kLog model (2)**

The declarative kLog model for the game is:

```
begin_domain.
signature sphere(sphere_id::self,color::property,weight::property)::extensional.
signature cube(cube_id::self,color::property,weight::property)::extensional.
signature next_s(sphere1_id::sphere,sphere2_id::sphere)::extensional.
signature next_c(cube1_id::cube,cube2_id::cube)::extensional.
signature next_sc(sphere1_id::sphere,cube1_id::cube)::extensional.
signature sph_eq_col(sphere1_id::sphere,sphere2_id::sphere)::extensional.
signature diff_shape_eq_weight(sphere_id::sphere,cube_id::cube)::extensional.
signature cub_eq_col(cube1_id::cube,cube2_id::cube)::intensional.
end_domain.
```

**KLog example: Declarative kLog model (1)**

- The E/R model can now be coded declaratively in kLog using an extension of the logic programming language Prolog
- Every entity or relation that will be used to generate features is declared with the keyword signature (similar to declarative bias used in PLP):
  - extensional signatures: all ground atoms have to be listed explicitly in the database, whereas intensional signatures implicitly define ground atoms using Prolog definite clauses (declarative feature construction)
  - Intensional signatures: are mostly used to introduce additional background knowledge on the domain in the learning process (key characteristics of kLog and most SRL systems).
- Then intensional predicates are grounded: process similar to materialization in databases, that is, the atoms implied by the background knowledge and the facts in the example are all computed using Prolog's deduction mechanism.

```c
=> extensionalized database
```

**KLog example: E/R modelling – graphicatization process (2)**

Graphicalizations for 3 instances of the game:

(The top figure depicts the graphicalization of the interpretation of previous Listing)
We introduce the relation function and It identifies pairs of neighborhoods of radius subgraphs, as:

Figure 2.7: NSPDK features. Top: instance with two vertices \((r,d)\).

A neighborhood has \(\text{neighborhood}(G)\) with roots at distance \(d\) where \(d(x, v)\) is the shortest-path distance between \(x\) and \(v\).

In a graph \(G\), the induced subgraph on a set of vertices \(W = \{w_1, \ldots, w_n\}\) is a graph that has \(W\) as vertex set and contains every edge of \(G\) whose endpoints are in \(W\).

A neighborhood \(N_r(G)\) is therefore a topological ball with center \(v\) and radius \(r\).

We introduce the relation \(R_{r,d}\) defined in terms of neighborhood subgraphs as:

\[ R_{r,d} = \{(N^r_v(G), N^u_v(G), G) : d(u, v) = d\} \]

It identifies pairs of neighborhoods of radius \(r\) whose roots are exactly at distance \(d\).

\[ \kappa_{r,d}(G, G') = \sum_{A, B \in \mathcal{G}} 1_{A = A'} \cdot 1_{B = B'} \]

where \(R_{r,d}^{-1}(G)\) indicates the multiset of all pairs of neighborhoods of radius \(r\) with roots at distance \(d\) that exist in \(G\), and where 1 denotes the indicator function and \(\cong\) the isomorphism between graphs.

**kLog example: Features generation (1)**

- Feature generation from the resulting graphs is performed by the Neighborhood Sub-graph Pairwise Distance Kernel (NSPDK) [Costa and De Grave, 2010], a particular type of graph kernel.
- NSPDK is a decomposition kernel in which pairs of sub-graphs are compared to each other in order to calculate the similarity between 2 graphs.
- These subgraphs (circles in the graph) are defined by 3 hyper-parameters.
  1. The root or center of the subgraph, « the kernel point », which can be any entity or relation in the graph, beforehand marked as a subset of the intensional and extensional domain relations.
  2. The radius \(r\) determines the size of the subgraphs, which entities or relations around the kernel point are taken into account.
  3. The distance \(d\) determines how far apart from each other the kernel points can be: each sub-graph around a kernel point that is within a distance \(d\) or less from the current kernel point will be considered.

- The kernel notion is finally given as the fraction of common fragments between 2 graphs.

**kLog example: Features generation (2)**

**Instance graph** with 2 vertices \(v, u\) selected as roots for neighborhood subgraphs at distance \(d = 2\):

**NSPDK features** or neighborhood pairs at distance \(d = 2\) with radius \(r = 0\) and 1 respectively (neighborhood subgraphs can overlap):

**kLog example: Kernels & SVM**

- A kernel is a similarity measure between 2 "observations", a dot product in a feature space.
- Kernels are used by SVMs (structural support vector machines) to efficiently compute a linear separator:

\[ k(x, x') = \langle \phi(x), \phi(x') \rangle \]

\[ \phi(.) \]

Input space

Feature space
kLog example: Graph Kernel definition (2)

Normalized version of $k_{r,d}$ is:

$$\hat{k}_{r,d}(G, G') = \frac{k_{r,d}(G, G')}{\sqrt{k_{r,d}(G, G)k_{r,d}(G', G')}}$$

To increase efficiency and generalization power, the zero-extension of $K$ is considered, obtained by imposing upper bounds on the radius and the distance parameter, with values to be determined e.g., via cross-validation:

$$K_{r_{\text{max}},d_{\text{max}}}(G, G') = \sum_{r=0}^{r_{\text{max}}} \sum_{d=0}^{d_{\text{max}}} \hat{k}_{r,d}(G, G')$$

The result is an extended high-dimensional feature space on which a statistical learning algorithm can be applied. Different learning tasks can be performed on this resulting feature space.

To this end, kLog interfaces with several solvers, including LibSVM [Chang and Lin, 2011] and SVM SGD [Bottou, 2010].

4. KLogNLP: KLog for NLP

- KLogNLP Definition & Workflow
- KLogNLP Declarative Model
- KLogNLP: an Example of Application


KLogNLP: Declarative model in kLog

The declarative kLogNLP model in kLog is:

```prolog
begin_domain.

signature sentence(sent_id::self)::extensional.
signature next(sent_id1::sentence, sent_id2::sentence)::extensional.

signature word(word_id::self, word_string::property, lemma::property, pos::property, namedEntity::property)::extensional.
signature next(word_id1::word, word_id2::word)::extensional.

signature corePhrase(coref_id::self)::extensional.
signature isPartOfCorePhrase(coref_phrase::corePhrase, word::word)::extensional.

signature coref(core_fphrase1::corePhrase, core_fphrase2::corePhrase)::extensional.

signature synonymous(word_id1::word, word_id2::word)::extensional.

signature dependency(word_id1::word, word_id2::word)::extensional.

kernel_points([word]).
end_domain.
```
kLogNLP: Problem considered (1)

Problem:

In a statistical relational learning approach perform the automatic identification of PICO categories in medical abstracts. PICO concepts are: primary Problem (P) or population, main intervention (I), main intervention Comparison (C), and Outcome of intervention (O).

PICO helps determining what terms are important in a query and therefore it helps building the query, which is sent to the search repositories.

Learning:

- The constructed feature space constructed by kLogNLP contains one feature vector per sentence.
- Any statistical learner can be used on the feature space constructed by kLog. Here is used HMM2 [Tsochantaridis et al., 2004], an implementation of SVM (structural support vector machines) for sequence tagging.

kLogNLP: Preprocessing step (1)

- The sentences have been preprocessed with:
  - a named entity tagger: BiogaphTA named entity module, which matches token sequences with entries in the UMLS database
  - a dependency parser: GENIA dependency parser GDep [Sagae&Tsujii 2007]
- This information can be represented as an Entity/Relationship (E/R) diagram of the problem under consideration (medical abstracts in this case):

The entities are the words and sentences in the medical abstracts

kLogNLP: Problem considered (2)

Dataset:

- NICTA-PIBOSO dataset [Kim et al. 2011] contains 1,000 medical abstracts
- If the abstract contains section headings (e.g. Background, Methodology, Results, etc.), it is considered to be structured. This information can be used as a feature in the model.
- The other abstracts are regarded unstructured.

Results:

<table>
<thead>
<tr>
<th>Label</th>
<th>CV/S-way</th>
<th>MBT</th>
<th>Kim et al.</th>
<th>kLog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>87.1</td>
<td>64.9</td>
<td>87.92</td>
<td>70.67</td>
</tr>
<tr>
<td>Intervention</td>
<td>48.0</td>
<td>6.9</td>
<td>48.08</td>
<td>21.39</td>
</tr>
<tr>
<td>Outcome</td>
<td>95.8</td>
<td>75.9</td>
<td>96.03</td>
<td>80.51</td>
</tr>
<tr>
<td>Population</td>
<td>70.9</td>
<td>21.4</td>
<td>63.88</td>
<td>43.15</td>
</tr>
<tr>
<td>Study Design</td>
<td>50.0</td>
<td>7.4</td>
<td>47.44</td>
<td>8.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5 Conclusions

As a general observation, it is important to note

1. kLog performs better on the majority of the individual classes and in overall performance for the 5-way setting, where we tried to classify the sentences

2. The other abstracts are regarded unstructured.

Sentence considered:

« Surgical excision of CNV may allow stabilisation or improvement of vision. »

Sentence interpretation:

interpretation z representing the sentence:

sentence(z4, 4)
hasCategory(z4, 'background')
w(z4_1, 'Surgical', 'Surgical', b-np, jj, 'O', 'O') hasWord(z4, w4_1)
dh(w4_1, w4_2, nmod)
nextW(w4_2, w4_1)
w(z4_2, 'excision', 'excision', i-np, nn, 'O', 'O') hasWord(z4, w4_2)
dh(w4_2, w4_5, sub)
nextW(w4_5, w4_2)
w(z4_3, 'of', 'of', b-pp, in, 'O', 'O') hasWord(z4, w4_3)
dh(w4_3, w4_2, nmod)
nextW(w4_2, w4_3)
w(z4_4, 'CNV', 'CNV', b-np, nn, 'B-protein', 'O') hasWord(z4, w4_4)
dh(w4_4, w4_3, pmid)
nextW(w4_3, w4_4)
...
kLogNLP: Declarative feature construction step (1)

- **kLog** permits to construct features *declaratively*, by using **intensional relations**.
- This permits to encode additional **background knowledge** based on a small set of preprocessed features, which renders experimentation very flexible and makes the results more interpretable.
- It furthermore allows one to limit the required features to the core discriminative ones.
- These **intensional features** are defined through definite clauses, and is done using an extension of the declarative programming language Prolog.

kLogNLP: Declarative feature construction step (2)

**Features for structured abstracts**: 4 intentional relations are used:
- Relation `lemmaRoot(S,L)` selects the lemmas of the root word in the dependency tree, which markedly limits the number of word features used:
  ```prolog
  lemmaRoot(S,L) :-
  hasWord(S, L), w(lam,_,_,_,_), dh(_, root).
  ```
- Relation `hasHeaderWord(S,X)` identifies whether a sentence is a header of a section. It selects the words of a sentence that count more than four characters (to discard short names of biological entities), which all need to be uppercase:
  ```prolog
  hasHeaderWord(S,X) :-
  w(W,X,_,_,_,_), hasWord(S,W),
  (atom(X) | name(X,C); C = X),
  length(C,Len), Len > 4, all_upper(C).
  ```
- Relation `hasSectionHeader(S,X)` : the sentences below a certain section header need to be marked as belonging to this section:
  ```prolog
  hasSectionHeader(S,X) :-
  next$(S1,S), hasHeaderWord(S1,X).
  ```
- Relation `nextS(S1,S), not isHeaderSentence(S), once(hasSectionHeader(S1,X)).`

kLogNLP: Graphicalization step (1)

- **Interpretations are graphicalized**, i.e. transformed into graphs.
- Since the facts that form the interpretation still conform to the E/R-diagram, this can be interpreted as unfolding the E/R-diagram over the data:

![Graphicalization Diagram](image-url)
kLogNLP: Features generation step (1)

- Feature generation uses the graphs of interpretation obtained in Graphicalization step
- Done by means of a graph kernel $K$, which calculates the similarity between 2 graphized interpretations
- Any graph kernel that allows fast computations on large graphs and has a flexible bias to enable heterogeneous features can in theory be applied.
- In the current implementation is used an extension of the Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) [Costa and De Grave, 2010] with 3 hyperparameters: kernel point, radius $r$ and distance $d$:

$$K(G,G') = \sum_{r=0}^{R} \sum_{d=0}^{D} k((A,B),(A',B'))$$

- The decomposition kernel is defined by relations $R_{r,d}$:
  $$k((A,B),(A',B')) = 1 \text{ iff (A,B) and (A',B') are pairs of isomorphic subgraphs — hard match kernel}$$
  $$k((A,B),(A',B')): \text{ multinomial distribution of labels in (A,B) or (A',B') — soft match kernel}$$

**Propositional learning setting**

Introduction to PILP: Bernard ESPINASSE 57

Introduction to PILP: Bernard ESPINASSE 58

Introduction to PILP: Bernard ESPINASSE 59

Introduction to PILP: Bernard ESPINASSE 60
kLogNLP: Learning step

- The constructed feature space constructed by kLogNLP contains one feature vector per sentence.
- Any statistical learner can be used on the feature space constructed by kLog.
- Here is used HMM2 [Tsochantaridis et al., 2004], an implementation of SVM (structural support vector machines) for sequence tagging.