PhD in Computer Science and Engineering
Bologna, April 2016

Machine Learning

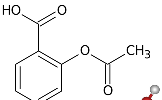
Marco Lippi

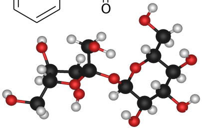
marco.lippi3@unibo.it

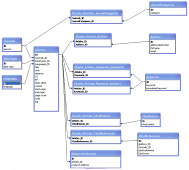
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

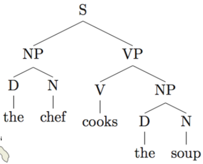
Statistical Relational Learning

In real-world problems, machine learning has very often to deal with highly **structured** and **relational** data.









Why do we need SRL ?

- Statistical learning assumes examples to be **independent and identically distributed** (i.i.d.)
- Traditional logic approaches (ILP) can handle relations but assume **no noise or uncertainty** in data

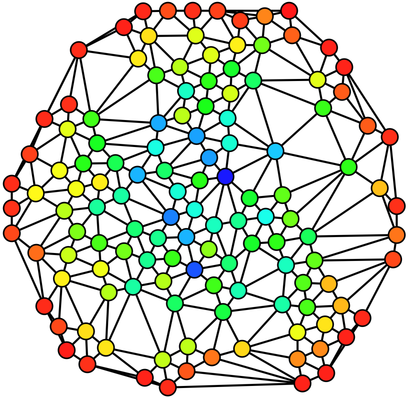
Many application domains need both **uncertainly handling** and **knowledge representation** !

Why do we need SRL ?

General observations

- Improve **perfomance** on certain tasks ☺
- Improve **interpretability** of results ☺
- Maybe computationally more **expensive** ☹
- Learning could be much **harder** ☹

Collective classification



Node classification

- User profiles in a social network
- Gene functions in a regulatory network
- Congestions in a transportation network
- Service requests in p2p networks
- Fault diagnosis in sensor networks
- Hypertext categorization on the Internet
- ...

Node classification

- use attributes of each node
- use attributes of neighbor nodes
- use features coming from the graph structure
- **use labels of other nodes**

In social networks, two important phenomena can happen

- **homophily** $\rightarrow$ a link between individuals is correlated with those individuals being similar in nature
- **co-citation regularity** $\rightarrow$ similar individuals tend to be related/connected to the same things

Structured and relational data: tasks

Node classification

Iterative approaches:

1. label a few nodes, given existing labels and features
2. train using the newly labeled examples
3. repeat the procedure until all nodes are labeled

Random walk approaches:

- the probability that a node v receives label y is that of a random walk starting at v will end at a node labeled with y
- could be solved by **label propagation** or **graph regularization**

Edge classification

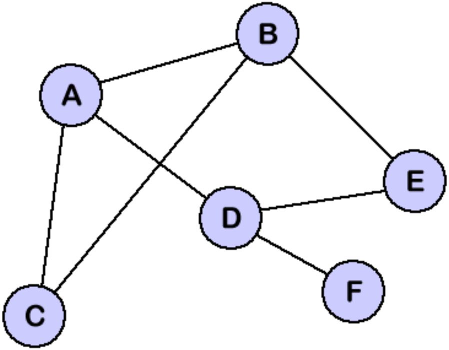
Classify properties of edges

- **viral marketing**: influence of a user on his neighbors
- **argumentation mining**: support/attack relationships
- **social networks**: types of interatcions between users

In social networks there are two main theories:

- **balance** $\rightarrow$ the friend of my friend is my friend, the enemy of my friend is my enemy
- **status** $\rightarrow$ positive edge between v_i and v_j means v_j has a higher status than v_i

Link prediction (or Link mining)



Link prediction

- Friendship in a social network
- Recommendation in a customer-product network
- Interaction in a biological network
- Link congestion in a transportation network
- Link congestion in a p2p network
- ...

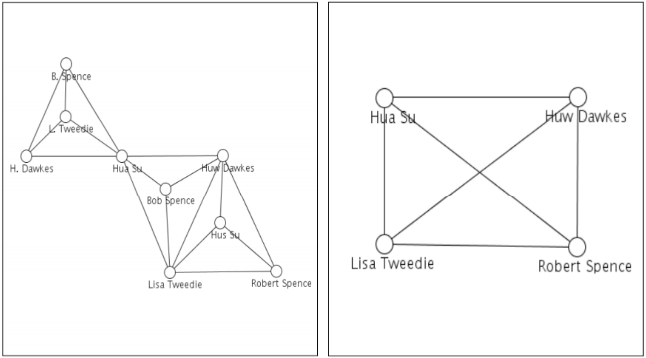
Structured and relational data: tasks

Networks are very often **dynamic**:

- nodes may change over time
- links may change over time
- node properties may change over time
- edge properties may change over time

Shall we predict the **evolution** of the network ?
E.g., given current links, which ones are likely to **change** ?

Entity resolution



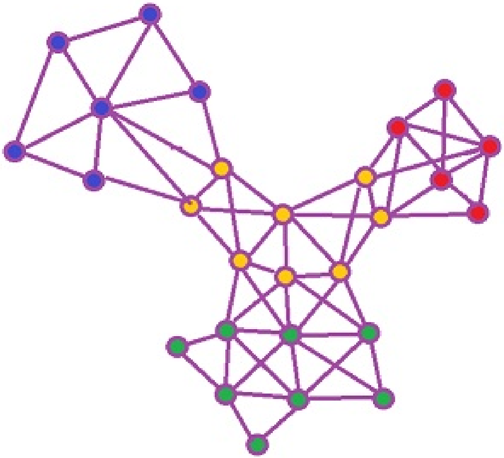
[Image from Entity Resolution Tutorial, VLDB2012, Lise Getoor]

Structured and relational data: tasks

Entity resolution

- Same paper in a bibliography corpus
- IP aliases over the Internet
- User matching across social networks
- Name disambiguation in a collection of documents
- Multiple records across databases

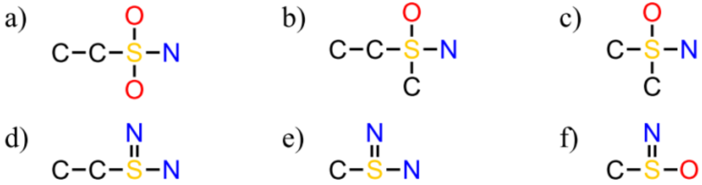
Group detection



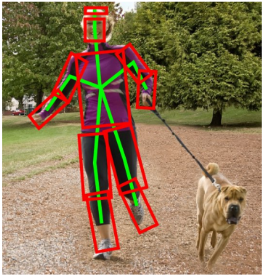
Group detection

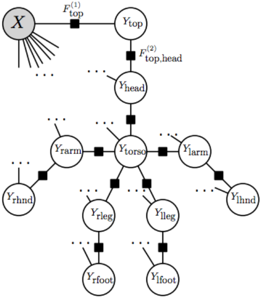
- Communities in mobile networks
- Terrorist cells in social networks
- Correlation in international flight databases
- Document ranking and hypertext connectivity
- Information retrieval and document clustering
- ...

Frequent subgraph discovery

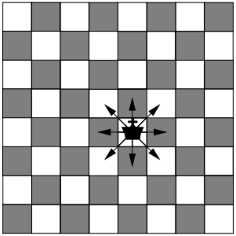


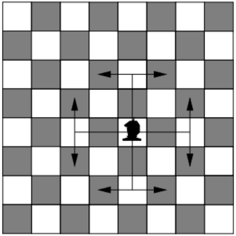
Structure learning





Predicate invention


a) The King


b) The Knight

[Image from Khan et al., ILP 1998]

The “relational revolution” of SRL stays at the intersection of Artificial Intelligence, Logic and Statistics

- **Logic**
  - $\rightarrow$ powerful and expressive formalism
  - $\rightarrow$ describes a domain in terms of quantified logic formulae
  - $\rightarrow$ allows to easily include background knowledge
- **Probabilistic graphical models**
  - $\rightarrow$ represent dependencies between random variables
  - $\rightarrow$ naturally handle uncertainty in data
  - $\rightarrow$ i.e., Bayesian Networks, Markov Random Fields

Propositional logic

- each **proposition** is associated to a symbol
- symbols have an associated **truth value** (true/false)
- logical operators (connectives) and inference at **propositional** level

Example

P = If it rains, Alice carries an umbrella
Q = It rains
R = Alice carries an umbrella

Can we **infer (entail)** R from P and Q ?

First-Order Logic (FOL)

- propositional logic is **not enough expressive**
- FOL described domains with objects, properties, relations
- it employs connectives as well as **quantifiers**

Constants: Alice, Italy, Snoopy, . . .
Variables: x, y, z, . . .
Functions: father_of, . . .
Predicates: blonde, friends, . . .
Atoms: blonde(Alice), friends(x,father_of(y))
Literals: blonde(Alice), ¬blonde(Alice)
Formulas: $\forall$ x,y,z friends(x,y) $\wedge$ friends(y,z) => friends(x,z)

Inference in first-order logic is **semi-decidable**

Remark

Semi-decidability of logic entailment $KB \models F$ implies that there is an algorithm which can always tell correctly when F is entailed by KB, but can provide either a negative answer or no answer at all in the case that F is not entailed by KB.

Inductive Logic Programming

A research area that started **during the 70s**

- Logic programming (Prolog)

Key ideas:

- describe the domain of interest in terms of **logic facts**
- **background knowledge** is also encoded in the logic database
- **learn concepts** (rules) directly from data
- try to **entail** all positive examples and none of the negatives

Example
FOIL – First Order Inductive Learner

- **General-to-specific** learner
- Starts from an empty clause
- Iteratively adds literals
- Exploits **information gain**

Example:
strawberry(x)
strawberry(x) :- red(x)
strawberry(x) :- red(x), smaller_than_apple(x)
strawberry(x) :- red(x), smaller_than_apple(x), bigger_than_ribes(x)

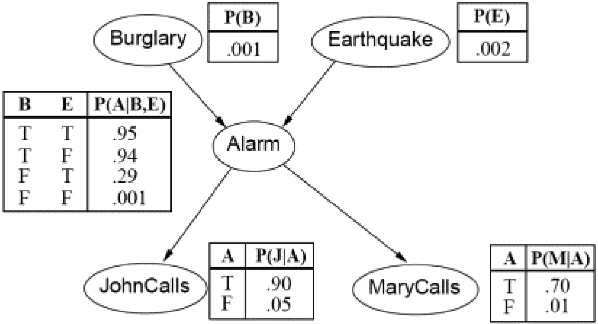
Graphical models

A graphical model is a **probabilistic model** where a graph encodes the **conditional dependence between random variables**

The model can be either directed or undirected:

- **directed** $\rightarrow$ Bayesian networks
- **undirected** $\rightarrow$ Markov networks

- The network structure is a **directed acyclic graph**
- It represents the **joint probability** of random variables
- Node have associated local **conditional probability tables**



[Image from Russel & Norvig]

Given variables $X_1, \ldots, X_n$, the joint probability is computed as:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(x_i | pa_i)$$

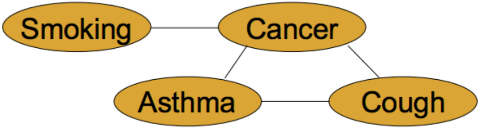
where pa_i are the parents of node i

Use **Bayes theorem** to find the values of other variables

Markov networks (or Markov random fields)

- The network structure is an **undirected graph**
- It represents the **joint probability** of random variables
- Random variables have to satisfy Markov properties
- Each clique has an associated **potential function**

$$P(X = x) = \frac{1}{Z} \prod_{C \in cl(G)} \phi_C(x_C)$$

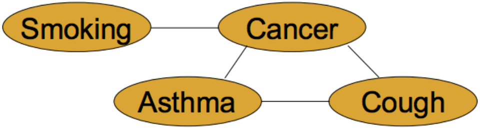


[Image from washington.edu]

Markov networks (or Markov random fields)

Markov properties:

1. Any two non-adjacent variables are conditionally independent given all other variables
2. A variable is conditionally independent of all other variables given its neighbors
3. Any two subsets of variables are conditionally independent given a separating subset



[Image from washington.edu]

Markov networks (or Markov random fields)

Any Markov network can be written as a **log-linear model**:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_k w_k^T f(x_{\{k\}}) \right)$$

Remark

A Markov network can represent dependencies which a Bayesian network cannot (e.g., cyclic dependencies) and the other way round (induced dependencies)

Inference in graphical models

Probabilistic inference: **inferring** the posterior distribution of **unobserved** variables, given observed ones

Exact methods (#P-complete):

- variable elimination
- clique tree propagation
- recursive conditioning
- ...

Approximate methods:

- loopy belief propagation
- Markov Chain Monte Carlo
- importance sampling
- ...

Example: **Markov Chain Monte Carlo with Gibbs sampling**

1. state $\leftarrow$ random truth assigment
2. for $i \leftarrow 1$ to numSamples
3.   for each variable x
4.   sample x according to $P(x|\text{neighbors}(x))$
5.   state $\leftarrow$ state with new value of x
6. $P(F) \leftarrow$ fraction of states where F is true

[Slide from washington.edu]

The SRL alphabet soup:

- Stochastic Logic Programs (SLPs) [Muggleton, 1996]
- Relational Bayesian Networks (RBNs) [Jaeger, 1997]
- Bayesian Logic Programs (BLPs) [Kersting & De Raedt, 2001]
- Probabilistic Relational Models (PRMs) [Friedman et al., 2001]
- Relational Dependency Networks (RDNs) [Neville & Jensen, 2007]
- Problog [De Raedt et al., 2007]
- Type Extension Trees (TETs) [Jaeger & al., 2013]
- Learning From Constraints (LFC) [Gori & al., 2014]
- **Markov Logic Networks** (MLNs) [Domingos & Richardson, 2006]
- . . .

Markov Logic Networks

Markov Logic Networks [Domingos & Richardson, 2006]

An MLN is a set of FOL formulae with attached weights

An example

```
1.2   Friends(x,y) ∧ WatchedMovie(x,m) => WatchedMovie(y,m)
2.3   Friends(x,y) ∧ Friends(y,z) => Friends(x,z)
0.8   LikedMovie(x,m) ∧ Friends(x,y) => LikedMovie(y,m)
...   ...
```

The **higher** the weight of a clause →

→ The **lower** the probability for a world violating that clause

What is a **world** or **Herbrand interpretation** ?

→ A truth assignment to all ground predicates

Markov Logic Networks

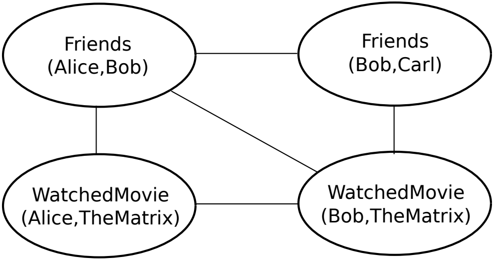
Together with a (finite) set of (unique and possibly typed) constants, an MLN defines a **Markov Network** which contains:

1. a binary node for each predicate grounding in the MLN, with value 0/1 if the atom is false/true
2. an edge between two nodes appearing together in (at least) one formula on the MLN
3. a feature for each formula grounding in the MLN, whose value is 0/1 if the formula is false/true, and whose weight is the weight of the formula

Set of constants

`people = {Alice,Bob,Carl,David}`

`movie = {BladeRunner,ForrestGump,PulpFiction,TheMatrix}`



The semantics of MLNs induces a probability distribution over all possible worlds. We indicate with X a set of random variables represented in the model, then we have:

$$P(X = x) = \frac{\exp\left(\sum_{F_i \in \mathcal{F}} w_i n_i(x)\right)}{Z}$$

being $n_i(x)$ the number of true groundings of formula i in world x.

The definition is similar to the joint probability distribution induced by a Markov network and expressed with a log-linear model:

$$P(X = x) = \frac{\exp\left(\sum_j w_j f_j(x)\right)}{Z}$$

Discriminative setting

- Typically, some atoms are always observed (**evidence X**), while others are unknown at prediction time (**query Y**)

EVIDENCE	QUERY
`Friends(Alice,Bob)`	`LikedMovie(Alice,BladeRunner)` ?
`Friends(Bob,Carl)`	`LikedMovie(Alice,PulpFiction)` ?
`WatchedMovie(Alice,PulpFiction)`	`LikedMovie(Bob,BladeRunner)` ?
`WatchedMovie(David,BladeRunner)`	`LikedMovie(Bob,TheMatrix)` ?
...	...

$$P(Y = y|X = x) = \frac{\exp\left(\sum_{F_i \in F_y} w_i n_i(x, y)\right)}{Z_x}$$

Discriminative setting

Three main MLN tasks:

1. Inference
2. Parameter Learning
3. Structure Learning

Popularity of Markov Logic:

$\rightarrow$ **Alchemy**, an open-source software

Inference

In the discriminative setting, inference corresponds to finding **the most likely interpretation** (MAP – Maximum A Posteriori)

$$y^* = \operatorname*{argmax}_y P(Y = y | X = x)$$

- **#P-complete** problem $\rightarrow$ **approximate** algorithms
- **MaxWalkSAT** [Kautz et al., 1996], stochastic local search
  $\rightarrow$ corresponds to minimizing the sum of weights of unsatisfied clauses

Note: there are also algorithms for estimating the **probability** of query atoms to be true

Inference: MaxWalkSAT [Kautz et al., 1996]

Main ideas:

- start with a **random** truth value assignment
- **flip** the atom giving the highest improvement (greedy)
- can get stuck in local minima
- sometimes perform a **random flip**
- stochastic algorithm (many runs often needed)
- need to build the whole **ground** network !

Main drawback

Memory explosion: with N constants and c as highest clause arity, the ground network requires $O(n^c)$ memory

One possible solution

Exploit sparseness by **lazily** grounding clauses (LazySAT) . . . But

still not enough !

Lifted inference

Key ideas:

- exploit **symmetries**
- reason at **first-order** level
- reason about **groups of objects**
- **scalable inference**

Inference

In many cases, one would prefer to have a **probability distribution** over query atoms, rather than just inferring the most likely state.

Several attempts:

- Markov Chain Monte Carlo over the Markov Network induced by the Markov Logic Networks, and the given set of constants
- Markov Chain SAT (MC-SAT), combining Markov Chain Monte Carlo and WalkSAT

Parameter Learning

Maximize the **conditional likelihood** of query predicates given evidence ones: requires **inference** as subroutine !

$$\frac{\partial}{\partial w_i} \log P(Y = y|X = x) = n_i - E_w[n_i]$$

Several different algorithms can be adopted to address the task:

- Voted Perceptron
- Contrastive Divergence
- Diagonal Newton
- (Preconditioned) Scaled Conjugate Gradient

Structure Learning

Directly **infer** the rules from the data !

A classic task for Inductive Logic Programming (ILP), which for MLNs can be addressed together with parameter learning

- Modified ILP algorithms (e.g., Aleph)
- Bottom-Up Clause Learning
- Iterated Local Search
- Structural Motifs

Still an open problem !

Modified ILP algorithms

Two-step approach:

1. learn a set of clauses with FOIL
2. just learn the weights with Alchemy

It works quite well in practice, but it still relies on a **non-probabilistic** ILP framework !

Jointly learning clauses and weights should be better !

Structure Learning

Bottom-Up Clause Learning, Iterated Local Search, Structural Motifs

- Start with **unit clauses** (single literal)
- Add/Remove literal
- Measure evaluation function (e.g. pseudo-likelihood)
- Learn weights by **counting groundings**

Hypertext Classification

```
Topic(page,topic)
HasWord(page,word)
Link(page,page)

HasWord(p,+w) => Topic(p,+t)
Topic(p,t) ∧ Link(p,q) => Topic(q,t)
```

Information Retrieval

```
InQuery(word)
HasWord(page,word)
Link(page,page)
Relevant(page)
```

```
HasWord(p,+w) ∧ InQuery(w) => Relevant(p)
Relevant(p) ∧ Link(p,q) => Relevant(q)
```

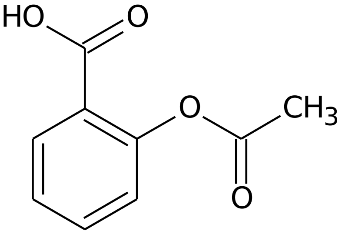
Entity Resolution

```
HasToken(token,field,record)
SameField(field,record,record)
SameRecord(record,record)
```

```
HasToken(+t,+f,r1) ∧ HasToken(+t,+f,r2) =>
SameField(f,r,s)
SameField(f,r1,r2) => SameRecord(r1,r2)
SameRecord(r1,r2) ∧ SameRecord(r2,r3) =>
SameRecord(r1,r3)
```

Applications: a chemical OCR

From images to structured data...



- `IsOxygen(A1)`
- `IsCarbon(A2)`
- `IsCarbon(A3)`
- `DoubleBond(A1,A2)`
- `SingleBond(A1,A3)`
- ...

http://mlocsr.dinfo.unifi.it

System Architecture

Two-stage architecture

1. Image Processing level
2. Markov Logic level

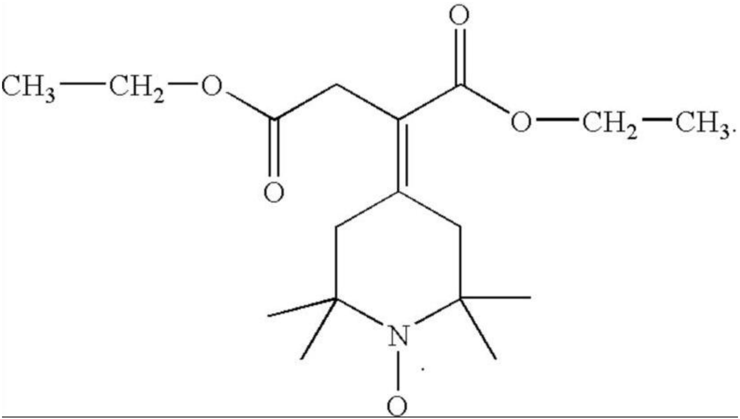
Main idea

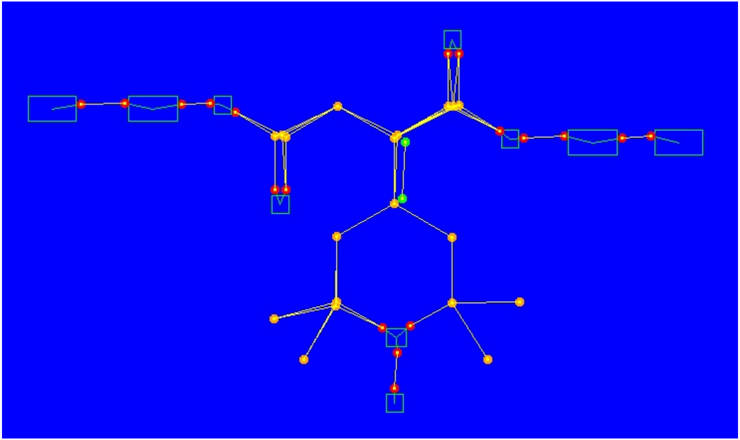
$\rightarrow$ extract low-level features

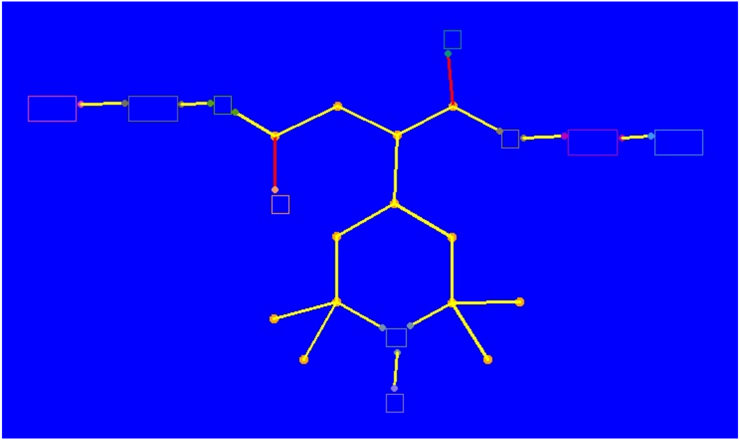
$\rightarrow$ translate them into Markov logic rules

$\rightarrow$ infer the final structure

Original input image







Opponent modeling in games

Games have always been a great opportunity for AI

- incomplete information
- uncertainty in data
- a lot of background knowledge

A nice domain also for SRL

Opponent modeling in games

Understand an adversarial's move given her past behavior:

- describe the domain in terms of **logic predicates**
- represent past played matches
- try to learn **strategies** through probabilistic rules

Handling continuous features

In many cases it is necessary to use **continuous features**

- classic machine learning classifiers naturally employ them
- they are often crucial in order to build accurate predictors

$\rightarrow$ How to integrate them with first-order logic ?

Markov logic networks with grounding-specific weights

Solution: grounding-specific weights [Lippi & Frasconi, 2009]

Instead of having a weight for a first-order logic rule, we allow different weights for different ground clauses.

Introducing vector of features

$$\texttt{Node(N12,\$Features\_N12)} \wedge \texttt{Node(N23,\$Features\_N23)}$$
$$\Rightarrow \texttt{Link(N12,N23)}$$

$$\texttt{Node(N18,\$Features\_N18)} \wedge \texttt{Node(N25,\$Features\_N25)}$$
$$\Rightarrow \texttt{Link(N18,N25)}$$

The two rules will have different weights, which will be computed using as feature vectors the "constants" with the dollar symbol ($).

Markov logic networks with grounding-specific weights

Re-parametrize the MLN by computing each weight w_i as a function of variables of each specific grounding c_{ij}:

Standard MLN

$$P(Y = y|X = x) = \frac{\exp\left(\sum_{F_i \in F_y} w_i n_i(x,y)\right)}{Z_x}$$

MLNs with grounding-specific weights

$$P(Y = y|X = x) = \frac{\exp\left(\sum_{F_i \in F_y} \sum_j w_i(c_{ij},\theta_i) n_{ij}(x,y)\right)}{Z_x}$$

Instead of having a weight for a first-order logic rule, we allow different weights for different ground clauses.

Markov logic networks with grounding-specific weights

The weights $w_i(c_{ij}, \theta_i)$ can be computed in several ways

- using **neural networks**, by taking as input an encoding of the grounding c_{ij} (in principle any feature can be used !)

- **Inference** algorithms do not change.
- **Learning** algorithm can implement gradient descent:

$$\frac{\partial P(y|x)}{\partial \theta_k} = \frac{\partial P(y|x)}{\partial w_i} \frac{\partial w_i}{\partial \theta_k}$$

where the **first** term is computed by MLN inference
and the **second** one is computed by backpropagation.

Other frameworks

Stochastic Logic Programs (SLPs) [Muggleton, 1996]

Generalization of:

- Hidden Markov Models
- Probablistic Context-Free Grammars

Based on the concept of **stochastic clause**:

$$\lambda \quad A \leftarrow B_1, \ldots, B_n$$

The weight λ represents the probability that a clause is used in a **derivation**, provided that **head is true**

If some rules have no λ, the SLP is called **impure**, and it can allow **both** stochastic and deterministic derivations

Directed acyclic graph:

- each node corresponds to a **relation** r in the domain
- attached **probability formula** $F_r(x_1, \ldots, x_n)$ over the symbols in the parent nodes of r

Semantics

For every relation r, the RBN associates a probability distribution over interpretations of r, which is defined on the interpretations of $pa(r)$ within the underlying Bayesian network

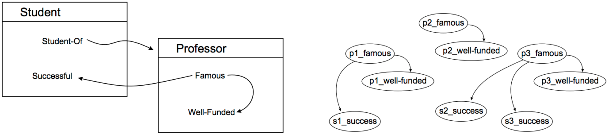
Probabilistic Relational Models (PRMs) [Friedman et al., 2001]

Probability distribution over a **relational database**:

- description of the relational schema of the domain
- probabilistic dependencies among its objects

Given a set of ground objects, a PRM specifies a **joint probability distribution** over all instantiations of a relational schema.

Probabilities on unseen variables, given **partial** instantiations



[Image from mit.edu]

Problog [De Raedt et al., 2007]

Probabilistic extension of Prolog

- logic clauses with associated **probability**
- probability of each clause is **independent** of the others
- distribution over **logic programs**

$$P(L|T) = \prod_{c_j \in L} p_j \prod_{c_j \notin L} (1 - p_j)$$

Initially designed to answer probabilistic queries

Type Extension Trees (TETs) [Jaeger & al., 2013]

Representation formalism for **complex combinatorial features** in relational domains.

A TET is a tree which represents a feature or a set of features:

- nodes contain **literals**, or conjunctions of literals
- edges are labeled with **sets of variables** (possibly empty)

TETs are syntactically closely related to predicate logic formulae, where subtrees correspond to sub-formulae.

TETs have been used for feature discovery and for classification

Combining kernel machines and first-order logic

Translate logic clauses (background knowledge) into a **semantic regularization** term, in addition to classic regularization

$$\min_f \sum_{i=1}^{N} V(f) + R(f) + S(f)$$

S(f) can encode **generic constraints**

- logic rules transformed through *p*-norms

Conclusions and future work

SRL aims at combining logic and statistical learning
$\rightarrow$ crucial when dealing with relational data

The problem is far for being solved:

- **scalability**
- **expressivity**
- more **efficient** approximation algorithms
- extend to many other **applications** and **contexts** !

. . . How about incorporating **deep learning** ?