kLog

A Language for Logical and Relational Learning with Kernels

Paolo Frasconi[1] **Fabrizio Costa**[2] **Luc De Raedt**[3] **Kurt De Grave**[3]

[1]Università di Firenze, Italy [2]Universität Freiburg, Germany [3]K.U. Leuven, Belgium

CoLISD.ECML/PKDD — Athens, 09.09.2011

- A language/framework for kernel-based relational learning

kLog

- A language/framework for kernel-based relational learning
- Currently embedded in Prolog

kLog

- A language/framework for kernel-based relational learning
- Currently embedded in Prolog
- Four simple concepts:
  - Learning from interpretations
  - Entity/relationship data modeling
  - Deductive databases
  - Graph kernels

Goals

- Make it possible to design and maintain complex features in a declarative fashion

Goals

- Make it possible to design and maintain complex features in a declarative fashion
- Make it possible to define in the same framework several kinds of learning problems ranging from plain classification and regression to entity classification to (hyper)link prediction and even unsupervised learning

Goals

- Make it possible to design and maintain complex features in a declarative fashion
- Make it possible to define in the same framework several kinds of learning problems ranging from plain classification and regression to entity classification to (hyper)link prediction and even unsupervised learning
- Modularity and separation of concerns:
  - Plug-in different graph kernels to create actual features
  - Plug-in different statistical, inference, optimization techniques
  - Simple semantics: the meaning of a kLog script only defines the learning problem and the associated features

Supervised learning: A quite general formulation

- Fit a linear *potential* function on some feature space:

$$F(x, y) = w'\phi(x, y)$$

where x and y are input and output ground atoms

Supervised learning: A quite general formulation

- Fit a linear *potential* function on some feature space:

$$F(x, y) = w'\phi(x, y)$$

  where x and y are input and output ground atoms
- $F(x, y)$ measures the *compatibility* between x and y

Supervised learning: A quite general formulation

- Fit a linear *potential* function on some feature space:

$$F(x, y) = w'\phi(x, y)$$

  where x and y are input and output ground atoms
- $F(x, y)$ measures the *compatibility* between x and y
- Predictions can be obtained as

$$f(x) = \arg\max_{y} F(x, y)$$

  This "inference" step is intractable in general (depending on the structure of the interdependencies between variables)

A classic trilogy

Propositional

Naïve Bayes

A classic trilogy

Propositional

Naïve Bayes

Logistic
regression

A classic trilogy

Propositional
Naïve Bayes
Logistic regression
SVM

A classic trilogy

Similar features

Propositional
Naïve Bayes
Logistic regression
SVM

A classic trilogy

Similar features

Propositional	Sequences
Naïve Bayes	HMM
Logistic regression	Linear-chain CRF
SVM	SVM-HMM

A classic trilogy

Similar features

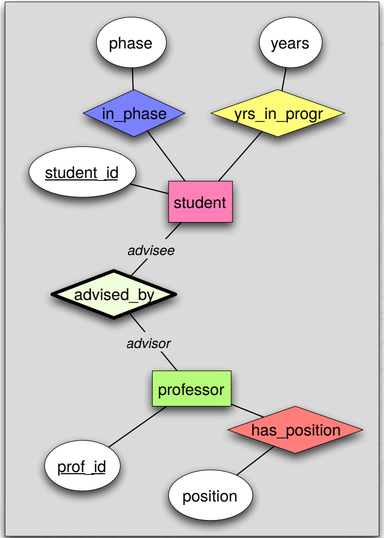
Propositional	Sequences	General relations
Naïve Bayes	HMM	Relational generative models, e.g. MLN, PRM
Logistic regression	Linear-chain CRF	Relational discriminative models, e.g. MLN, CRF
SVM	SVM-HMM	Max-margin relational machines, e.g. M^3N

A classic trilogy

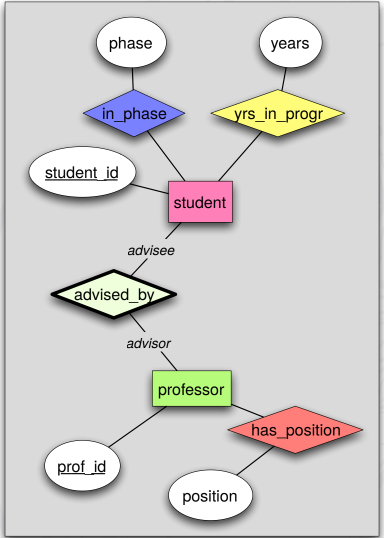
	Propositional	Sequences	General relations
Similar features	Naïve Bayes	HMM	Relational generative models, e.g. MLN, PRM
	Logistic regression	Linear-chain CRF	Relational discriminative models, e.g. MLN, CRF
	SVM	SVM-HMM	Max-margin relational machines, e.g. M^3N

Similar loss

kLog by example: UW-CSE

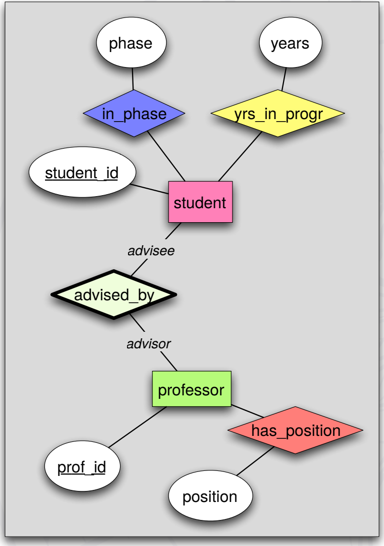


kLog by example: UW-CSE



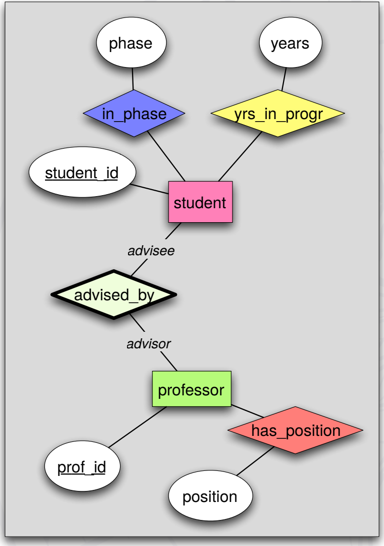
- Classic E/R diagram
- Boxes are entities
- Diamonds are relationships
- Ovals are properties
- Underlined properties are entity identifiers (not directly used to create features)

kLog by example: UW-CSE




```
signature student(
    student_id::self
)::extensional.
signature in_phase(
    student_id::student,
    phase::property)::extensional.
signature years_in_program(
    student_id::student,
    years::property
)::extensional.
signature professor(
  prof_id::self
)::extensional.
signature has_position(
    prof_id::professor,
    position::property
)::extensional.
```

kLog by example: UW-CSE




```
signature student(
    student_id::self
)::extensional.
signature in_phase(
    student_id::student,
    phase::property)::extensional.
signature years_in_program(
    student_id::student,
    years::property
)::extensional.
signature professor(
   prof_id::self
)::extensional.
signature has_position(
    prof_id::professor,
    position::property
)::extensional.

signature advised_by(
    student_id::student,
    prof_id::professor
)::extensional.
```

Data

- Data is a set of interpretations (in the pure logical sense)
- One interpretation is a set of ground facts

Data

- Data is a set of interpretations (in the pure logical sense)
- One interpretation is a set of ground facts
- Interpretations are independent
- In UW-CSE there is one interpretation for every research group (AI, Graphics, etc.)

Data

- Data is a set of interpretations (in the pure logical sense)
- One interpretation is a set of ground facts
- Interpretations are independent
- In UW-CSE there is one interpretation for every research group (AI, Graphics, etc.)
- Interpretations are *invisible* at the level of kLog scripts (since they are independent, you are not allowed to create interactions)

Data

- Data is a set of interpretations (in the pure logical sense)
- One interpretation is a set of ground facts
- Interpretations are independent
- In UW-CSE there is one interpretation for every research group (AI, Graphics, etc.)
- Interpretations are *invisible* at the level of kLog scripts (since they are independent, you are not allowed to create interactions)
- The keyword *extensional* declares that all true ground facts for the given predicate are actually given as data (under the usual CWA).

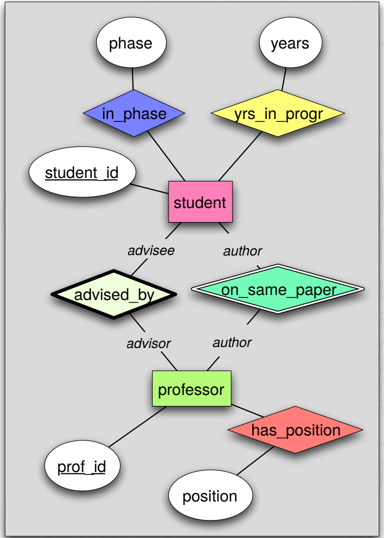
Example of interpretation

```
interpretation(ai,student(person311)).
interpretation(ai,student(person14)).
...
interpretation(ai,professor(person7)).
interpretation(ai,professor(person185)).
...
interpretation(ai,has_position(person292,faculty_affiliate)).
interpretation(ai,has_position(person79,faculty)).
...
interpretation(ai,in_phase(person139,post_quals)).
interpretation(ai,in_phase(person333,pre_quals)).
...
interpretation(ai,years_in_program(person382,year_3)).
interpretation(ai,years_in_program(person333,year_2)).
...
interpretation(ai,advised_by(person265,person168)).
interpretation(ai,advised_by(person352,person415)).
...
```

Interpretations may contain more relations

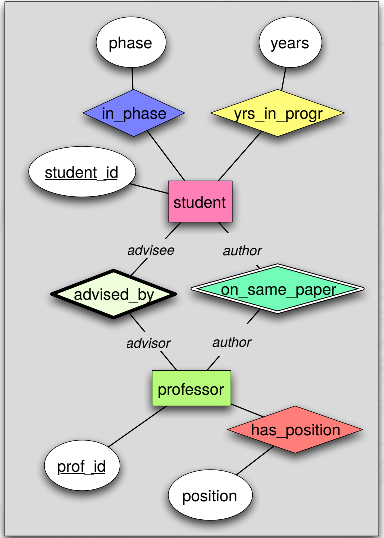
```
interpretation(ai,publication(title25,person284)).
interpretation(ai,publication(title284,person14)).
interpretation(ai,publication(title110,person14)).
...
interpretation(ai,taught_by(course12,person211,autumn_0001)).
interpretation(ai,taught_by(course123,person150,autumn_0001)).
interpretation(ai,taught_by(course44,person293,winter_0001)).
...
interpretation(ai,ta(course44,person193,winter_0304)).
interpretation(ai,ta(course128,person271,winter_0304)).
interpretation(ai,ta(course128,person392,winter_0304)).
```

Intensional signatures



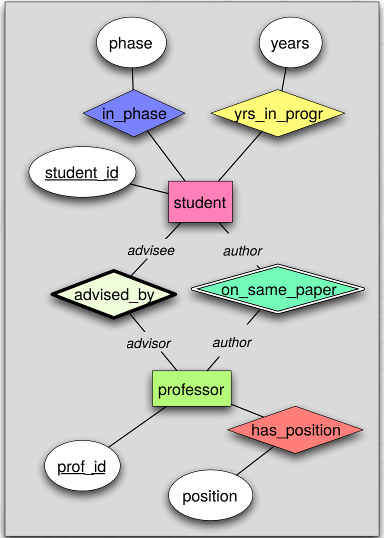

```
signature on_same_paper(
    student_id::student,
    prof_id::professor
)::intensional.
```

Intensional signatures




```
signature on_same_paper(
    student_id::student,
    prof_id::professor
)::intensional.

on_same_paper(S,P) :-
     student(S), professor(P),
     publication(Pub, S),
     publication(Pub,P).
```

Intensional signatures




```
signature on_same_paper(
   student_id::student,
   prof_id::professor
)::intensional.

on_same_paper(S,P) :-
    student(S), professor(P),
    publication(Pub, S),
    publication(Pub,P).

signature on_same_course(
  student_id::student,
  prof_id::professor
)::intensional.
on_same_course(S,P) :-
    professor(P), student(S),
    ta(Course,S,Term),
    taught_by(Course,P,Term).
```

Graphicalization

- Second semantic layer: map each interpretation into a simple graph (not a hypergraph).

Graphicalization

- Second semantic layer: map each interpretation into a simple graph (not a hypergraph).
- The mapping is lossless:
  - There is one vertex for every ground fact, labeled by the fact itself

Graphicalization

- Second semantic layer: map each interpretation into a simple graph (not a hypergraph).
- The mapping is lossless:
  - There is one vertex for every ground fact, labeled by the fact itself
  - One (undirected) edge between u and v iff:

    1) u is an entity-fact
    2) v is a relationship-fact
    3) v refers to the identifier in u

  (so the graph is bipartite)

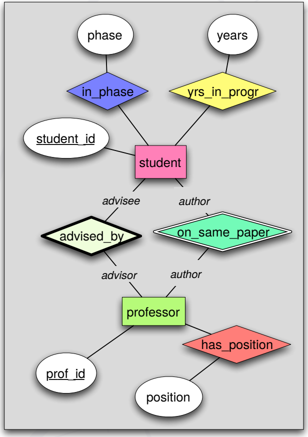
Graphicalization

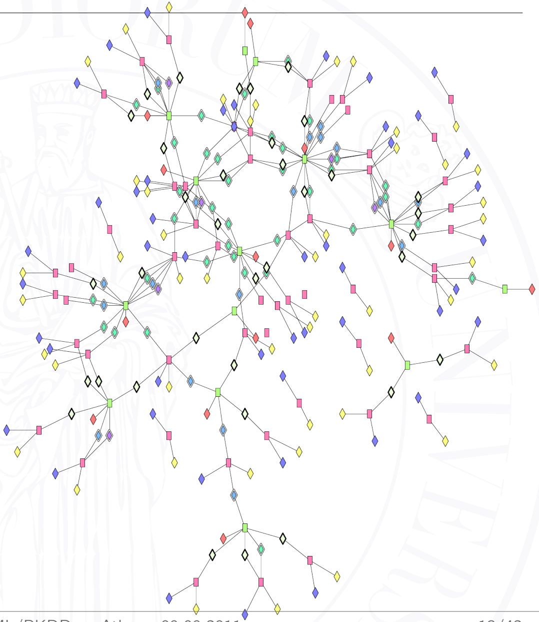
- Second semantic layer: map each interpretation into a simple graph (not a hypergraph).
- The mapping is lossless:
  - There is one vertex for every ground fact, labeled by the fact itself
  - One (undirected) edge between u and v iff:

    1) u is an entity-fact
    2) v is a relationship-fact
    3) v refers to the identifier in u

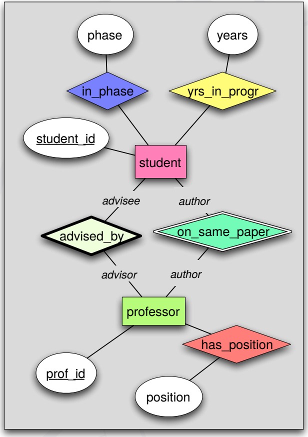
    (so the graph is bipartite)
- Graphicalization is essentially grounding the E/R diagram

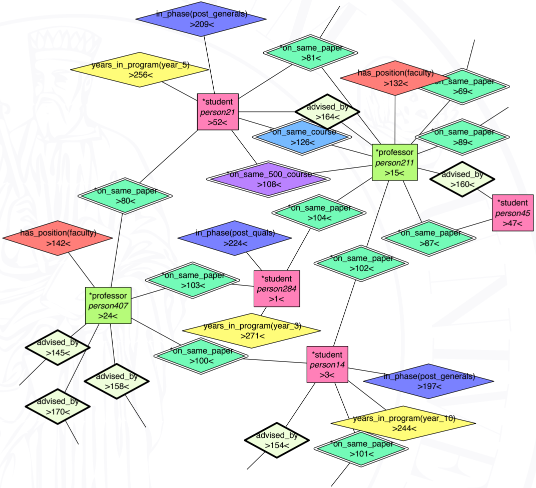
Graphicalization in UW-CSE





Graphicalization in UW-CSE





Supervised learning jobs

- A supervised learning job is defined by marking some signature(s) as target (aka query, aka output)

Supervised learning jobs

- A supervised learning job is defined by marking some signature(s) as target (aka query, aka output)
- This means to kLog: learn a statistical model capable of predicting tuples for the corresponding relation(s)

Supervised learning jobs

- A supervised learning job is defined by marking some signature(s) as target (aka query, aka output)
- This means to kLog: learn a statistical model capable of predicting tuples for the corresponding relation(s)
- The focus of a job is on what, not on how (different statistical models can solve the same job – maybe with different performances)

Single task binary classification

- Job type obtained when we specify a single target signature with **no properties**.
- Example:

```
signature advised_by(
  s::student,
  p::professor
)::extensional.
```

- In this example y consists of all ground atoms of the relation `advised_by`
- Each tuple of identifiers in the target relation, e.g. a (`student`,`professor`) pair, is called a case (or instance)
- if the target signature has entity sets $\mathcal{E}_1, \cdots, \mathcal{E}_k$, the set of cases is their the Cartesian product

Other jobs types

- Single-task multiclass classification: if the target signature contains a categorical property

```
signature pageclass(
  url::webpage,
  category::property
)::extensional.
```

Other jobs types

- Single-task multiclass classification: if the target signature contains a categorical property

```
signature pageclass(
  url::webpage,
  category::property
)::extensional.
```

- Single-task regression: if the target signature contains a numerical property

```
signature intelligence(
  student_it::student,
  qi::property
)::extensional.
```

Other jobs types

- Single-task multiclass classification: if the target signature contains a categorical property

```
signature pageclass(
  url::webpage,
  category::property
)::extensional.
```

- Single-task regression: if the target signature contains a numerical property

```
signature intelligence(
  student_it::student,
  qi::property
)::extensional.
```

- Multi-task: if there are several target signatures or a single target signature with several properties.

Overview of job types

# of properties	Relational arity 0	1	2
0	Binary classification of interpretations	Binary classification of entities	Link prediction
1	Multiclass / regression on interpretations	Multiclass / regression on entities	Attributed link prediction
>1	Multitask on interpretations	Multitask predictions on entities	Multitask attributed link prediction

Features (via graph kernels)

- The next semantic layer of kLog concerns feature vectors
- Here we use graph kernels

Features (via graph kernels)

- The next semantic layer of kLog concerns feature vectors
- Here we use graph kernels
- In principle any graph kernel can be used and the architecture of kLog is open enough to allow different feature generators to be plugged in

Features (via graph kernels)

- The next semantic layer of kLog concerns feature vectors
- Here we use graph kernels
- In principle any graph kernel can be used and the architecture of kLog is open enough to allow different feature generators to be plugged in
- In practice we have implemented a generalization of the Neighborhood Subgraph Pairwise Distance Kernel (NSPDK, Costa & De Grave, ICML 2010)

NSPDK: Core ideas

- Decompose graphs into subgraphs rooted at certain designated vertices called kernel-points (KP)
- Consider all pairs of such subgraphs
- Count the number of common pairs between two graphs
- Use hashing to approximate subgraph isomorphism

The NP-relation

- Given graph $G = (V, E)$ and $u, v \in V$, let

$$\delta_{u,v} = \begin{cases} \mathrm{SPD}(u, v) & \textit{if}\, u, v \in \mathrm{KP} \\ \infty & \text{otherwise} \end{cases}$$

where SPD is for shortest-path-distance

The NP-relation

- Given graph $G = (V, E)$ and $u, v \in V$, let

$$\delta_{u,v} = \begin{cases} \mathrm{SPD}(u, v) & \mathit{if}\, u, v \in \mathrm{KP} \\ \infty & \text{otherwise} \end{cases}$$

  where SPD is for shortest-path-distance
- Let $\mathcal{N}_r^v(G)$ denote the subgraph of G induced by all $x \in V$ s.t. $\mathrm{SPD}(x, v) \leq r$

The NP-relation

- Given graph $G = (V, E)$ and $u, v \in V$, let

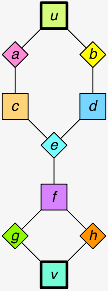
$$\delta_{u,v} = \begin{cases} \mathrm{SPD}(u, v) & \mathit{if}\, u, v \in \mathrm{KP} \\ \infty & \text{otherwise} \end{cases}$$

  where SPD is for shortest-path-distance
- Let $\mathcal{N}_r^v(G)$ denote the subgraph of G induced by all $x \in V$ s.t. $\mathrm{SPD}(x, v) \leq r$
- The neighborhood-pair (NP) relation is the set of triplets

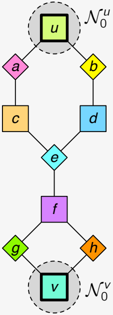
$$R_{r,d} = \{(A, B, G) : A \cong \mathcal{N}_r^v(G), B \cong \mathcal{N}_r^u(G), \delta_{u,v} = d\}$$

  where $\cong$ is graph isomorphism

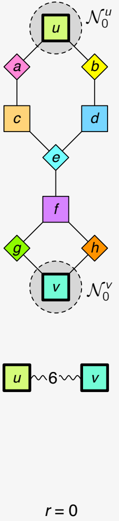
NP's for fixed $d = 6$



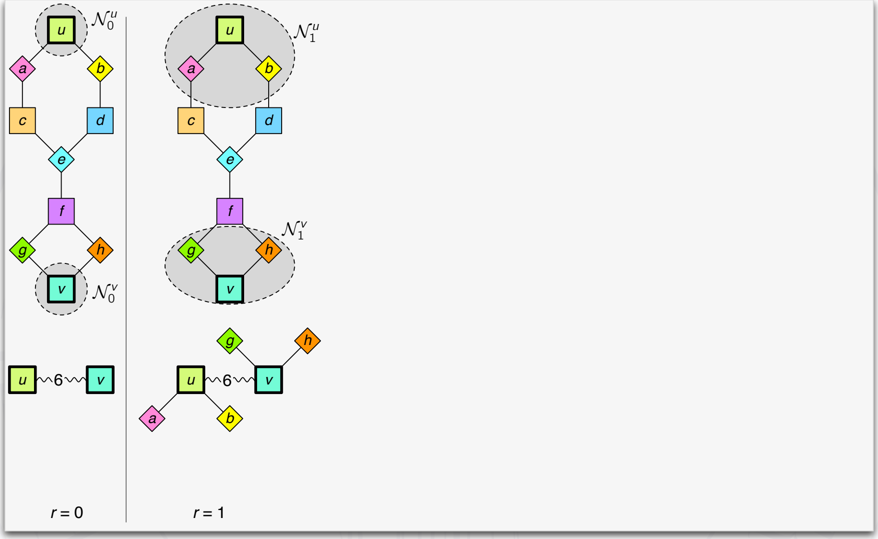
NP's for fixed $d = 6$



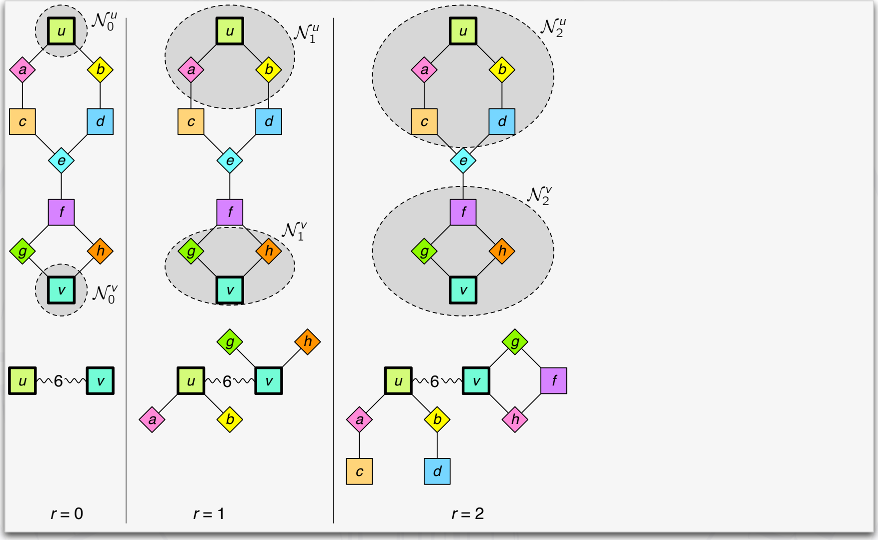
NP's for fixed $d = 6$



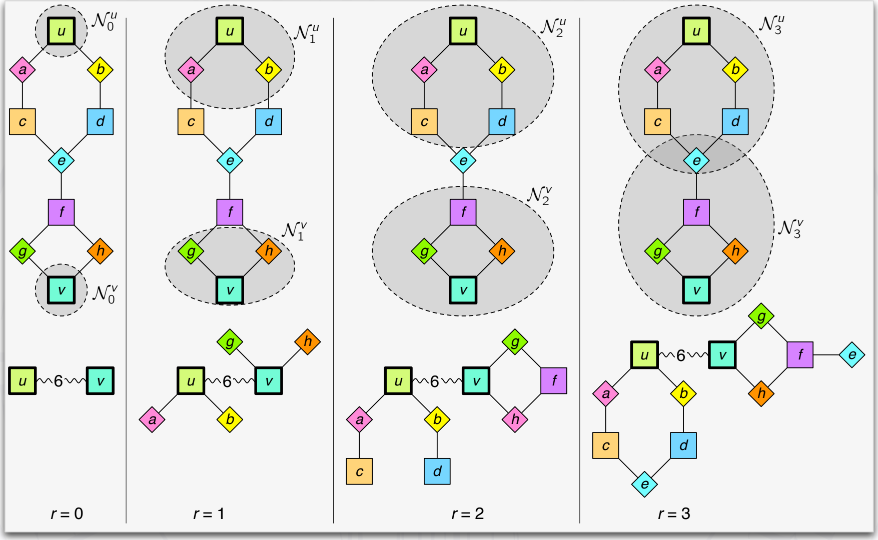
NP's for fixed $d = 6$



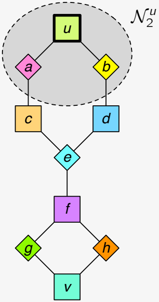
NP's for fixed $d = 6$



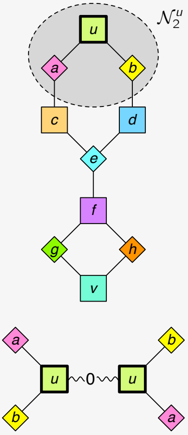
NP's for fixed $d = 6$



NP's for fixed $r = 2$

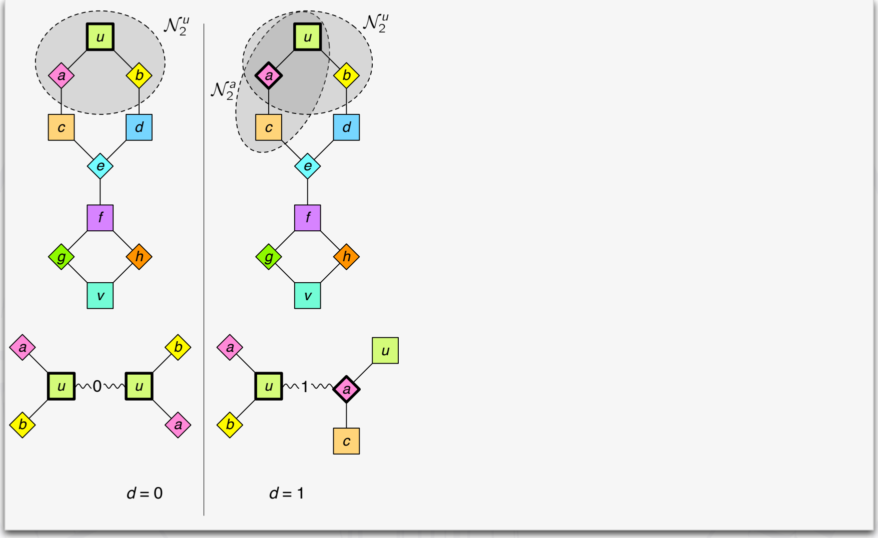


NP's for fixed $r = 2$

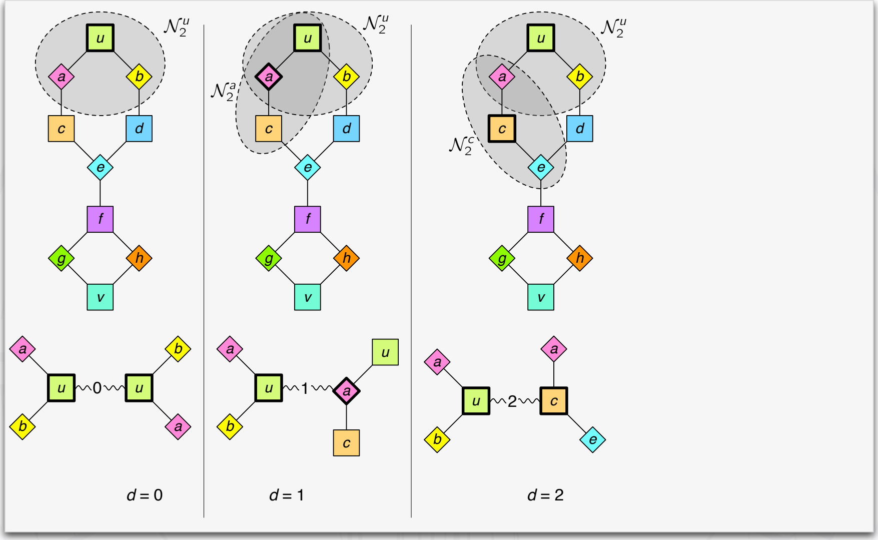


$d = 0$

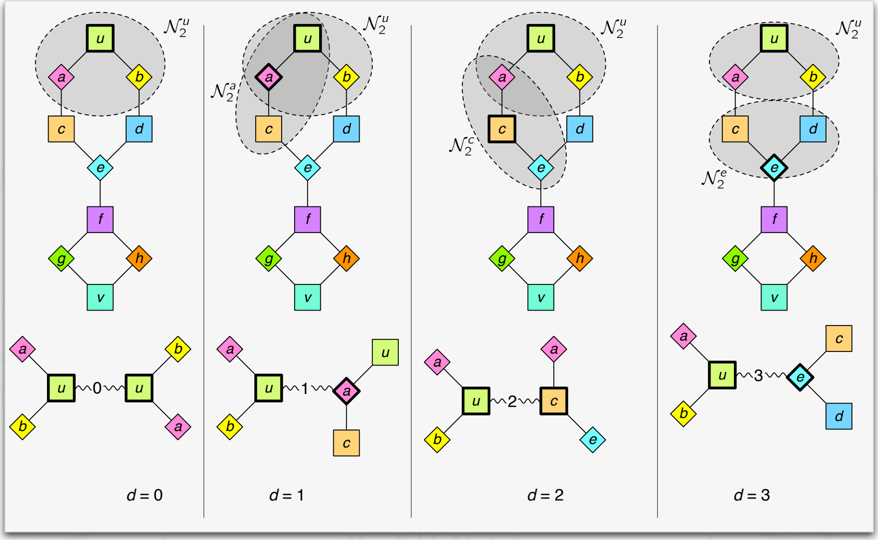
NP's for fixed $r = 2$



NP's for fixed $r = 2$



NP's for fixed $r = 2$



Definition of the NSPDK

- $\kappa_{r,d}$ counts common NP's between two graphs:

$$\kappa_{r,d}(G, G') = \sum_{\substack{(A,B) \in R^{-1}_{r,d}(G) \\ (A',B') \in R^{-1}_{r,d}(G')}} \delta(A, A')\delta(B, B')$$

where the "inverse" of a relation $R \subset \mathcal{A} \times \mathcal{B} \times \mathcal{C}$ is the multiset $R^{-1}(c) = \{(a, b) : R(a, b, c)\}$

Definition of the NSPDK

- $\kappa_{r,d}$ counts common NP's between two graphs:

$$\kappa_{r,d}(G, G') = \sum_{\substack{(A,B) \in R^{-1}_{r,d}(G) \\ (A',B') \in R^{-1}_{r,d}(G')}} \delta(A, A')\delta(B, B')$$

  where the "inverse" of a relation $R \subset \mathcal{A} \times \mathcal{B} \times \mathcal{C}$ is the multiset $R^{-1}(c) = \{(a, b) : R(a, b, c)\}$

- Overall kernel:

$$K(G, G') = \sum_{r=0}^{R} \sum_{d=0}^{D} \kappa_{r,d}(G, G').$$

Soft matches

- The hard-match might produce very “rare” features depending on the structure of the graph

Soft matches

- The hard-match might produce very “rare” features depending on the structure of the graph
- Bad if vertices have high degree

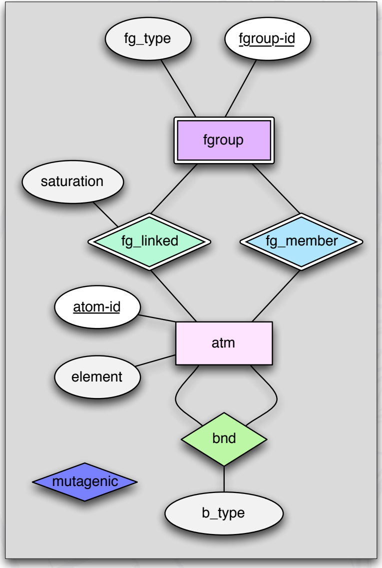
Soft matches

- The hard-match might produce very "rare" features depending on the structure of the graph
- Bad if vertices have high degree
- Soft match kernel:

$$\kappa_{r,d}(G, G') = \sum_{\substack{(A,B) \in R^{-1}_{r,d}(G) \\ (A',B') \in R^{-1}_{r,d}(G')}} \sum_{\substack{v \in V(A) \cup V(B) \\ v' \in V(A') \cup V(B')}} \delta(\mathcal{L}(v), \mathcal{L}(v'))$$

where $\mathcal{L}(v)$ is the label of vertex v

Example: small molecules




```
signature atm(
    atom_id::self,
    element::property)::extensional.
signature bnd(
    atom_1@b::atm,
    atom_1@b::atm,
    type::property)::extensional.

signature fgroup(
    fgroup_id::self,
    group_type::property
    )::intensional.
signature fgmember(
    fg::fgroup,
    atom::atm)::intensional.
signature fg_linked(
    fg::fgroup,
    alichain::fgroup,
    saturation::property)::intensional.

signature mutagenic::extensional.
```

A whole kLog script

```
:- use_module('klog').
begin_domain.
signature atm(atom_id::self,element::property)::extensional.
...
signature activity(act::property)::extensional.
kernel_points([atm,fgroup]).
end_domain.

experiment :-
   new_feature_generator(my_fg,nspdk),
   set_klog_flag(my_fg,radius,4),
   set_klog_flag(my_fg,distance,8),
   attach(bursi_ext),
   new_model(my_model,libsvm_c_svc),
   set_klog_flag(my_model,c,0.5),
   stratified_kfold(mutagenic,10,my_model,my_fg,muta_stratum).
```

Small molecules (regression/classification)

Biodegradability

Setting	RMSE	SCC	MAPE
Functional groups	1.07 ± 0.01	0.54 ± 0.01	14.0 ± 0.1
Atom bonds	1.13 ± 0.01	0.48 ± 0.01	14.5 ± 0.1

Small molecules (regression/classification)

Biodegradability

Setting	RMSE	SCC	MAPE
Functional groups	1.07 ± 0.01	0.54 ± 0.01	14.0 ± 0.1
Atom bonds	1.13 ± 0.01	0.48 ± 0.01	14.5 ± 0.1

Bursi

Setting	AUROC	F1	Error%
Functional groups	0.91 ± 0.01	86.78 ± 1.05	14.7 ± 1.5
Atom bonds	0.90 ± 0.01	85.21 ± 1.37	16.9 ± 1.5

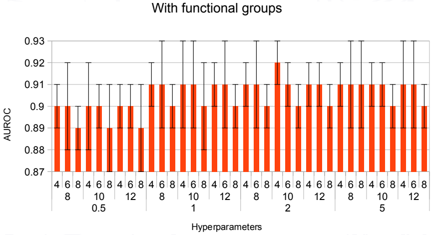
Small molecules (regression/classification)

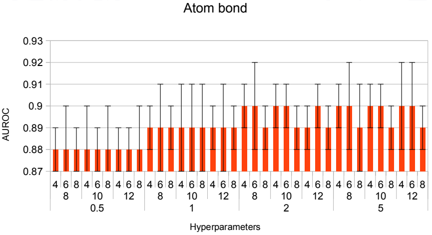
Biodegradability

Setting	RMSE	SCC	MAPE
Functional groups	1.07 ± 0.01	0.54 ± 0.01	14.0 ± 0.1
Atom bonds	1.13 ± 0.01	0.48 ± 0.01	14.5 ± 0.1

Bursi

Setting	AUROC	F1	Error%
Functional groups	0.91 ± 0.01	86.78 ± 1.05	14.7 ± 1.5
Atom bonds	0.90 ± 0.01	85.21 ± 1.37	16.9 ± 1.5





Is the NPDK kernel general enough?

- What about multiple interdependent predictions within the same interpretation?

Is the NPDK kernel general enough?

- What about multiple interdependent predictions within the same interpretation?
- Two possible answers:

Is the NPDK kernel general enough?

- What about multiple interdependent predictions within the same interpretation?
- Two possible answers:
  - The graph kernel creates joint features $\phi(x, y)$ so go for collective (structured-output) prediction i.e. argmax $w'\phi(x, y)$ over an exponential number of assignments to y. This is expecially challenging because intensional predicates need to be re-evaluated for different assignments.

Is the NPDK kernel general enough?

- What about multiple interdependent predictions within the same interpretation?
- Two possible answers:
  - The graph kernel creates joint features $\phi(x, y)$ so go for collective (structured-output) prediction i.e. argmax $w'\phi(x, y)$ over an exponential number of assignments to y. This is expecially challenging because intensional predicates need to be re-evaluated for different assignments.
  - Project the collective problem into several i.i.d. views

Viewpoints and i.i.d. views (non-collective)

- Let $c \in y$ be a case

Viewpoints and i.i.d. views (non-collective)

- Let $c \in y$ be a case
- The *viewpoint* of c, W_c, is the set of vertices that touch c in the graph

Viewpoints and i.i.d. views (non-collective)

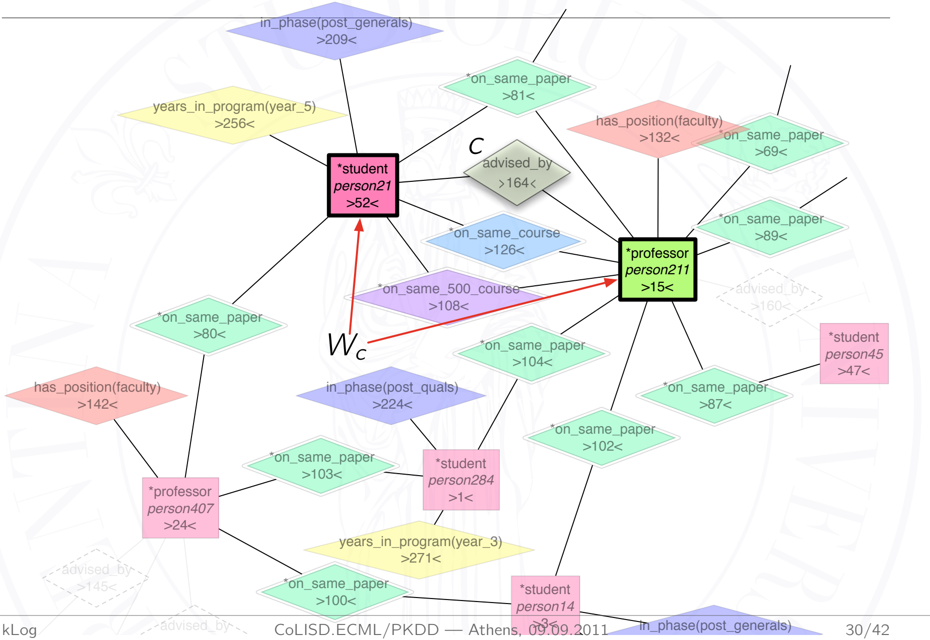
- Let $c \in y$ be a case
- The *viewpoint* of c, W_c, is the set of vertices that touch c in the graph
- Consider the mutilated graph G_c where all vertices in y except c are removed

Viewpoints and i.i.d. views (non-collective)

- Let $c \in y$ be a case
- The *viewpoint* of c, W_c, is the set of vertices that touch c in the graph
- Consider the mutilated graph G_c where all vertices in y except c are removed
- Define a kernel $\hat{\kappa}$ on mutilated graphs: like NSPDK but with the restriction that the first endpoint must be in W_c

$$\hat{R}_{r,d} = \{(A, B, G_c) : A \cong \mathcal{N}_r^v, B \cong \mathcal{N}_r^u, v \in W_c, \delta_{u,v} = d\}$$

Viewpoints example



View points and i.i.d. views (non-collective)

- We get in this way a kernel "centered" around case c:

$$\hat{K}(G_c, G'_{c'}) = \sum_{r,d} \sum_{\substack{A,B \in \hat{R}^{-1}_{r,d}(G_c) \\ A',B' \in \hat{R}^{-1}_{r,d}(G'_{c'})}} \delta(A, A')\delta(B, B')$$

View points and i.i.d. views (non-collective)

- We get in this way a kernel "centered" around case c:

$$\hat{K}(G_c, G'_{c'}) = \sum_{r,d} \sum_{\substack{A,B \in \hat{R}^{-1}_{r,d}(G_c) \\ A',B' \in \hat{R}^{-1}_{r,d}(G'_{c'})}} \delta(A, A')\delta(B, B')$$

- Finally let

$$K(G, G') = \sum_{c \in y, c' \in y'} \hat{K}(G_c, G'_{c'})$$

View points and i.i.d. views (non-collective)

- We get in this way a kernel "centered" around case c:

$$\hat{K}(G_c, G'_{c'}) = \sum_{r,d} \sum_{\substack{A, B \in \hat{R}^{-1}_{r,d}(G_c) \\ A', B' \in \hat{R}^{-1}_{r,d}(G'_{c'})}} \delta(A, A')\delta(B, B')$$

- Finally let

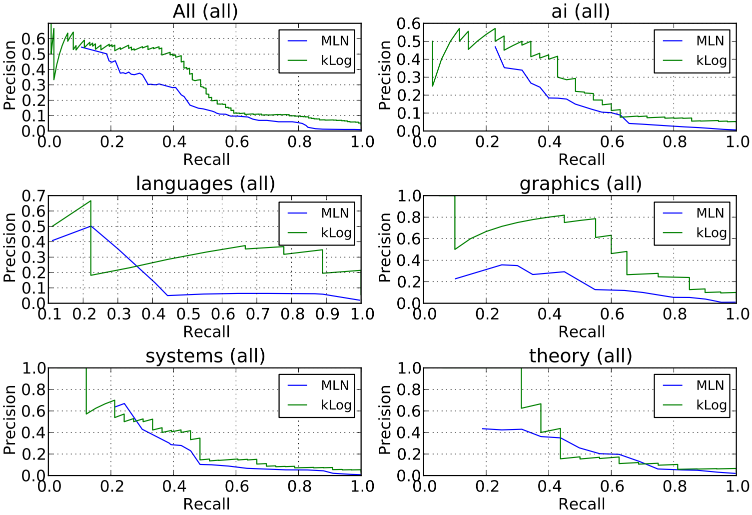
$$K(G, G') = \sum_{c \in y, c' \in y'} \hat{K}(G_c, G'_{c'})$$

- This kernel corresponds to the potential

$$F(x, y) = w' \sum_c \hat{\phi}(x, c)$$

which is clearly maximized by maximizing, independently, all sub-potentials $w'\hat{\phi}(x, c)$ with respect to c.

UW-CSE: All information



Stacking

- Alternative setting: we only know about persons without knowing whether they are professors or students

Stacking

- Alternative setting: we only know about persons without knowing whether they are professors or students
- Without collective inference we also lack the ability of joint inference on the predicates `student`, `professor`, and `advised_by`

Stacking

- Alternative setting: we only know about persons without knowing whether they are professors or students
- Without collective inference we also lack the ability of joint inference on the predicates `student`, `professor`, and `advised_by`
- In kLog it is easy to define a stacked (pipelined) prediction method:

Stacking

- Alternative setting: we only know about persons without knowing whether they are professors or students
- Without collective inference we also lack the ability of joint inference on the predicates `student`, `professor`, and `advised_by`
- In kLog it is easy to define a stacked (pipelined) prediction method:
  - First, learn to discriminate between professors and students

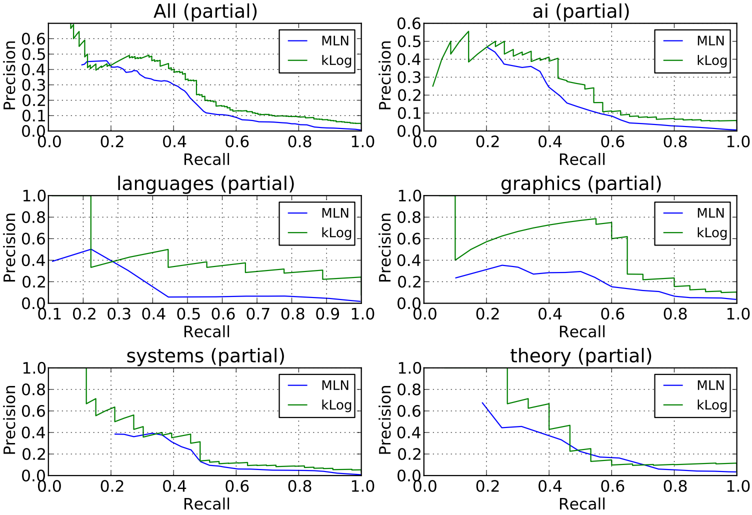
Stacking

- Alternative setting: we only know about persons without knowing whether they are professors or students
- Without collective inference we also lack the ability of joint inference on the predicates `student`, `professor`, and `advised_by`
- In kLog it is easy to define a stacked (pipelined) prediction method:
  - First, learn to discriminate between professors and students
  - Assert *induced* groundings (predicted in cross-validation mode)

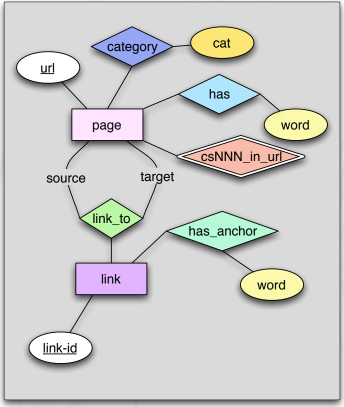
Stacking

- Alternative setting: we only know about persons without knowing whether they are professors or students
- Without collective inference we also lack the ability of joint inference on the predicates `student`, `professor`, and `advised_by`
- In kLog it is easy to define a stacked (pipelined) prediction method:
  - First, learn to discriminate between professors and students
  - Assert *induced* groundings (predicted in cross-validation mode)
  - Learn the binary relation taking saved groundings as additional data

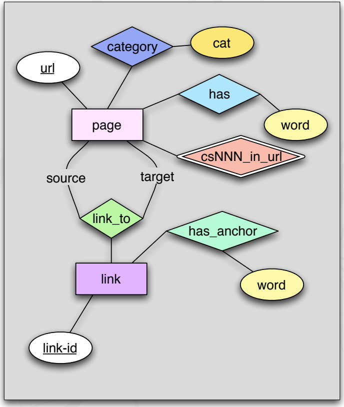
UW-CSE: Partial information



WebKB


category
cat
url
has
page
word
csNNN_in_url
source
target
link_to
has_anchor
link
word
link-id

WebKB

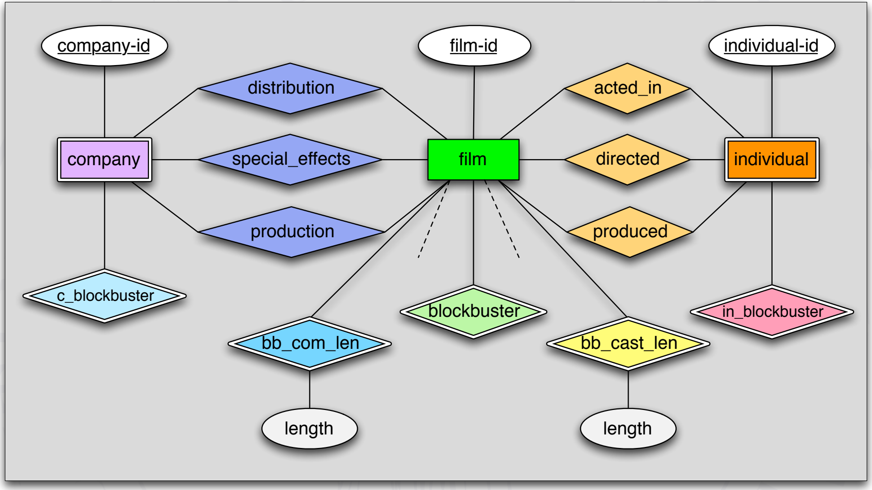



```
...
signature csNNN_in_url(
    pageid::page
)::intensional.

csNNN_in_url(Url) :-
    page(Url),
    atom_codes(Url,CUrl),
    regexp("cs(e*)[0-9]+",CUrl,[],[_Match]).

signature category(
    page_id::page,
    cat::property
)::extensional.

...
```

WebKB: results

```
# Cases:  1039
Case error rate:            12.42%
Interpretation error rate: 100.00%
Contingency table: (rows are predictions)
         researc faculty  course student
researc    59      11       4       15    a,p,r,f1= 0.94 0.66 0.70 0.68
faculty     9     125       2       50    a,p,r,f1= 0.91 0.67 0.82 0.74
 course     0       0     233        0    a,p,r,f1= 0.99 1.00 0.95 0.98
student    16      17       5      493    a,p,r,f1= 0.90 0.93 0.88 0.91
Average p,r,f1 = 0.89 0.88 0.88
```

Internet Movies Database



IMDb

```
signature blockbuster(film_id::film)::intensional.
blockbuster(M) :-
    opening_weekend(M,Receipts),
    Receipts > 2000000.
```

IMDb

```
signature blockbuster(film_id::film)::intensional.
blockbuster(M) :-
    opening_weekend(M,Receipts),
    Receipts > 2000000.

signature individual(individual_id::self)::intensional.
individual(P) :-
    person(P,_Name), active_enough(P).
has_active_role(P,M) :- acted_in(P,M).
has_active_role(P,M) :- directed(P,M).
has_active_role(P,M) :- produced(P,M).
active_enough(P) :-
    setof(M,has_active_role(P,M),Ms), length(Ms,N), N>2.
```

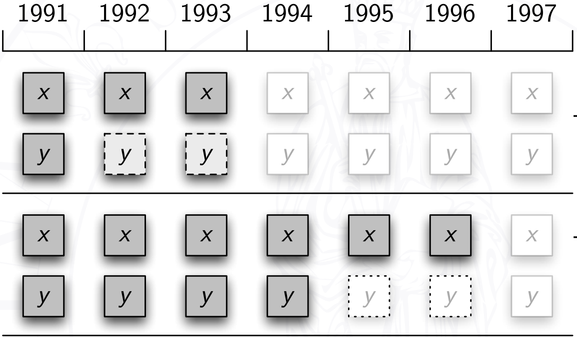
IMDb

```
signature blockbuster(film_id::film)::intensional.
blockbuster(M) :-
    opening_weekend(M,Receipts),
    Receipts > 2000000.

signature individual(individual_id::self)::intensional.
individual(P) :-
    person(P,_Name), active_enough(P).
has_active_role(P,M) :- acted_in(P,M).
has_active_role(P,M) :- directed(P,M).
has_active_role(P,M) :- produced(P,M).
active_enough(P) :-
    setof(M,has_active_role(P,M),Ms), length(Ms,N), N>2.

signature in_blockbuster(individual_id::individual)::intensional.
in_blockbuster(P) :-
    has_active_role(P,M),
    blockbuster(M).
signature bb_cast_len(film_id::film,n::property)::intensional.
bb_cast_len(M,N) :-
    setof(Actor, (acted_in(Actor,M), in_blockbuster(Actor)), Set),
    length(Set,N).
```

Slicing

- In a data set like IMDb there is one single interpretation
- How to split training and test data?

Slicing



Train on {1992,1993}

Test on {1995,1996}

= invisible = visible = targets = to be predicted

IMDb: results

Year	movies	facts	AUROC
1995	74	2483	-
1996	223	6406	-
1997	311	8031	0.85
1998	332	7822	0.92
1999	348	7842	0.88
2000	381	8531	0.95
2001	363	8443	0.94
2002	370	8691	0.93
2003	343	7626	0.94
2004	371	8850	0.94
2005	388	9093	0.92
All			0.92 ± 0.03

Conclusions

- Highlights
  - Complex feature generation thanks to graph kernels
  - Easy but powerful declaration of jobs
  - Most statistical learners pluggable-in (even as external programs)

Conclusions

- Highlights
  - Complex feature generation thanks to graph kernels
  - Easy but powerful declaration of jobs
  - Most statistical learners pluggable-in (even as external programs)
- To be done (or to be tried)
  - Collective classification (e.g. label propagation, MaxWalkSAT, ...)
  - Constraints
  - Multitask regularization

Conclusions

- Highlights
  - Complex feature generation thanks to graph kernels
  - Easy but powerful declaration of jobs
  - Most statistical learners pluggable-in (even as external programs)
- To be done (or to be tried)
  - Collective classification (e.g. label propagation, MaxWalkSAT, ...)
  - Constraints
  - Multitask regularization
- Applications (work in progress)
  - Information extraction from text, e.g. spatial role labeling (Kordjamshidi et al. 2011)
  - Hedge cue detection, e.g. recognition of weasel sentences (Verbeke et al. 2011)
  - Vision, e.g. image segmentation/labeling (Antanas et al. 2011)