UNIVERSITÀ
DEGLI STUDI
FIRENZE

DINFO
DIPARTIMENTO DI
INGEGNERIA
DELL'INFORMAZIONE

Kernel methods for structured data

Paolo Frasconi
Università degli Studi di Firenze, Italy

Summer school on Mining Big and Complex Data
Ohrid, September 4th, 2016

Outline

1. Learning with structured data
2. Kernels and convolution kernels
3. Graph kernels
4. Kernel methods for relational learning
5. Dealing with continuous/high dimensional attributes

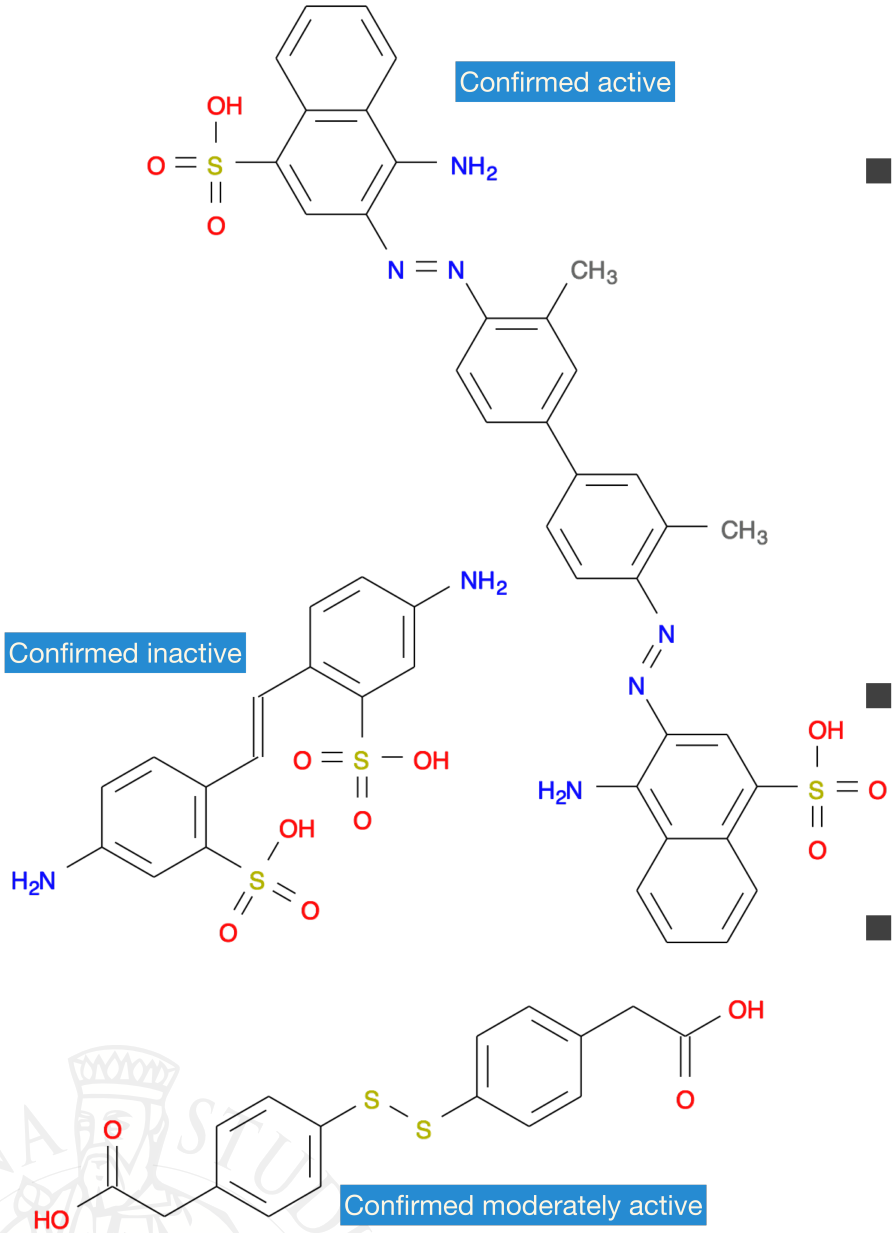
Part I

Learning with structured data

- Data in propositional supervised learning:
  - Design matrix (input) X: one example per row, one attribute (or codebit) per column
  - Target vector (output) y: one scalar per example (binary, regression, multitask)

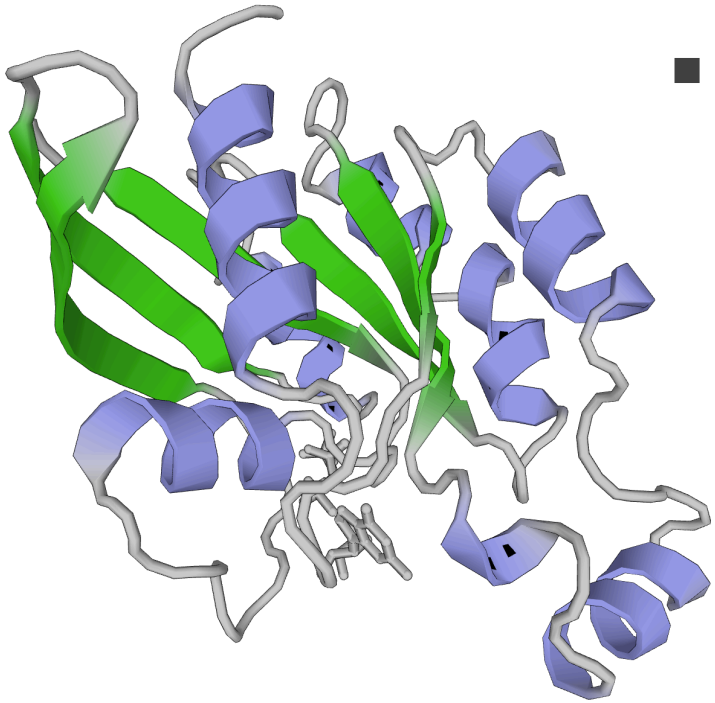
- Data in propositional supervised learning:
  - Design matrix (input) X: one example per row, one attribute (or codebit) per column
  - Target vector (output) y: one scalar per example (binary, regression, multitask)
- Data in relational learning:
  - Relational database(s) or equivalent (restricted) first-order logic representations — e.g. learning from interpretations (De Raedt 2008)

Example: Chemoinformatics



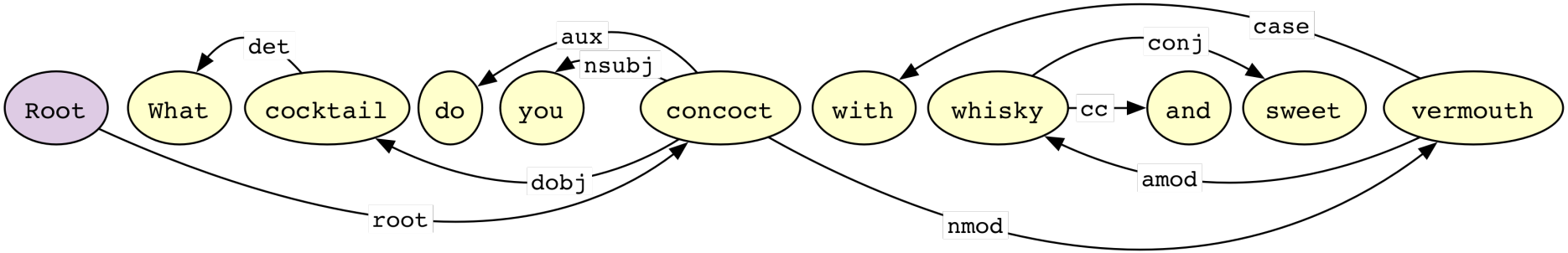
- Each molecule represented as a graph where:
  - Nodes correspond to atoms
  - Edges correspond to bonds
- Attributes may include element, charge, bond type
- Task: many are possible — e.g. active compound in drug design, mutagenicity, biodegradability, etc.

Example: Protein function (Borgwardt et al. 2005b)



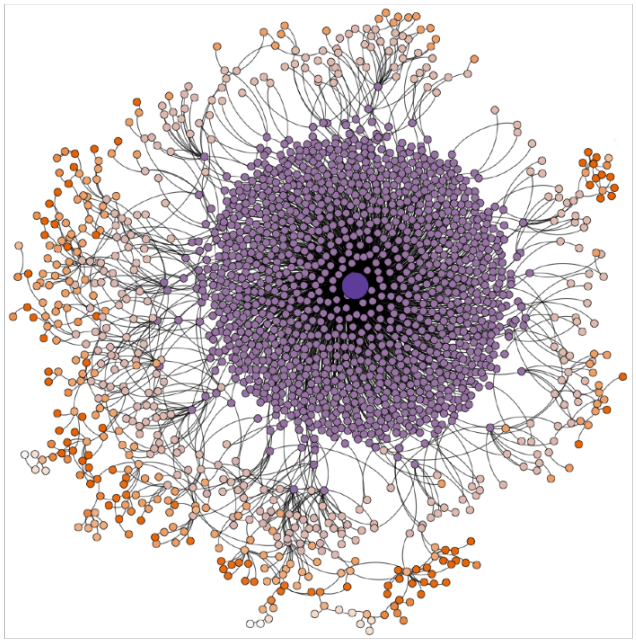
- Each protein represented as a graph where:
  - Nodes correspond to secondary structure elements (SSE)
  - Structural edges (SSE are neighbors in space) and sequential edges (SSE are adjacent in sequence)
- Attributes include physical and chemical information
- Task: discriminate between enzymes and non-enzymes, or categorize according to enzyme type

Example: Sentence classification



- Each sentence represented as a graph where:
  - Nodes correspond to words (possibly represented by word vectors)
  - Edges correspond to the (typed) dependency relation between governors and dependents
- Tasks: many are possible — e.g. classify sentences according to the expected answer, such as *food* (Li et al. 2006); detect weasel sentences (Verbeke et al. 2011), segment scientific abstracts (Verbeke et al. 2012) etc.

Example: sub-community identification (Yanardag et al. 2015)



- Each discussion on reddit represented as a graph where:
  - Nodes correspond to users
  - Edges represent user interactions (e.g. responding to each other's comments)
- Task: Categorize discussions into communities, e.g. question/answer-based community or a discussion-based community

Structured output learning

- You have a structured output learning problem when the output (target) is a data structure

Structured output learning

- You have a structured output learning problem when the output (target) is a data structure
- In general you can expect interdependencies among output variables

Structured output learning

- You have a structured output learning problem when the output (target) is a data structure
- In general you can expect interdependencies among output variables
- Thus better accuracy can be achieved by forming features that include these output variables

Example: Supervised sequence learning

- Example: protein secondary structure prediction
  - Input is a sequence of amino-acids
  - Target is a corresponding sequence of α-β-γ labels

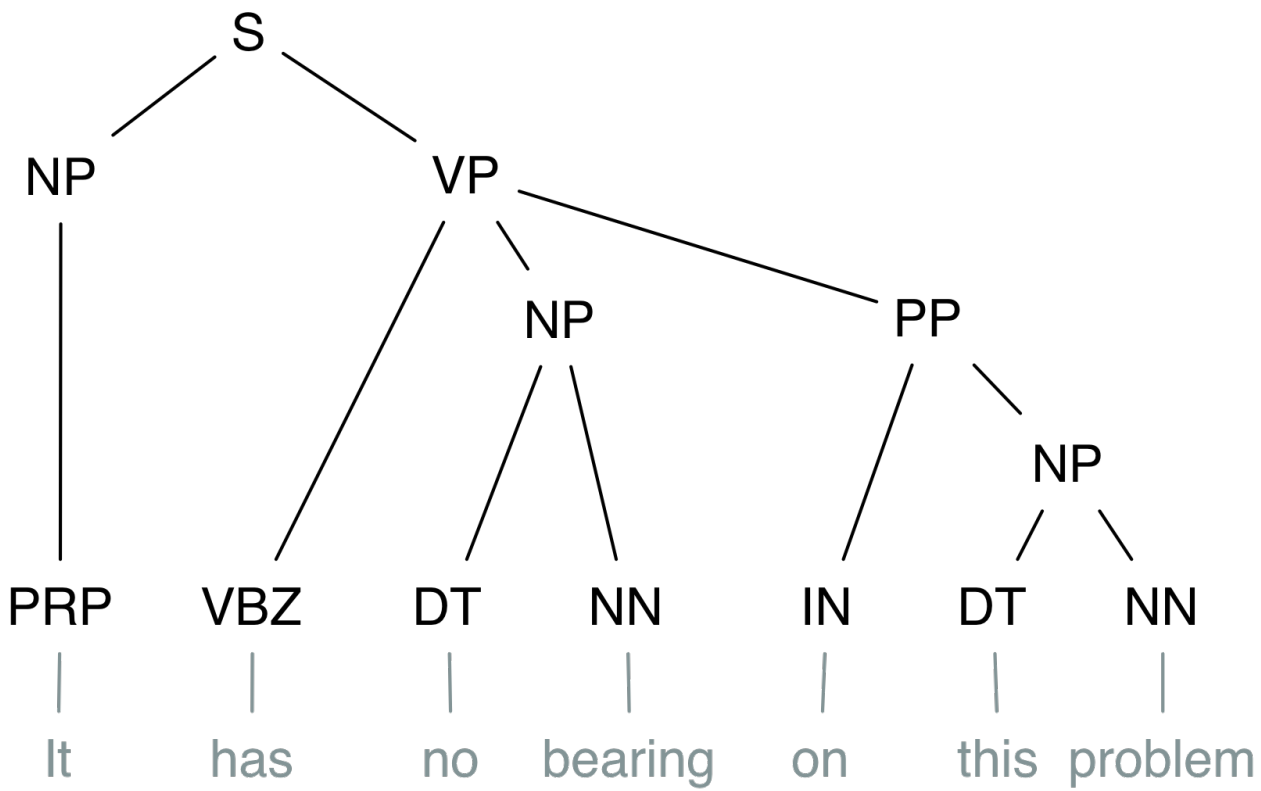
Example: Supervised sequence learning

- Example: protein secondary structure prediction
  - Input is a sequence of amino-acids
  - Target is a corresponding sequence of α-β-γ labels
- Example: Part-of-speech tagging
  - Input is a sentence (sequence of words)
    - Output is a corresponding sequence of syntactic categories

PRP	VBZ	DT	NN	IN	DT	NN
It	has	no	bearing	on	this	problem

Example: Natural language parsing

- Each example corresponds to a sentence
- The input is a sequence of words
  - The target is a parse tree for the sentence



Part 2

Kernels and convolution kernels

Basic material only, for details see e.g. (Haussler 1999; Schölkopf et al. 2002; Shawe-Taylor et al. 2004)

- Let $\mathcal{X}$ denote the instance space

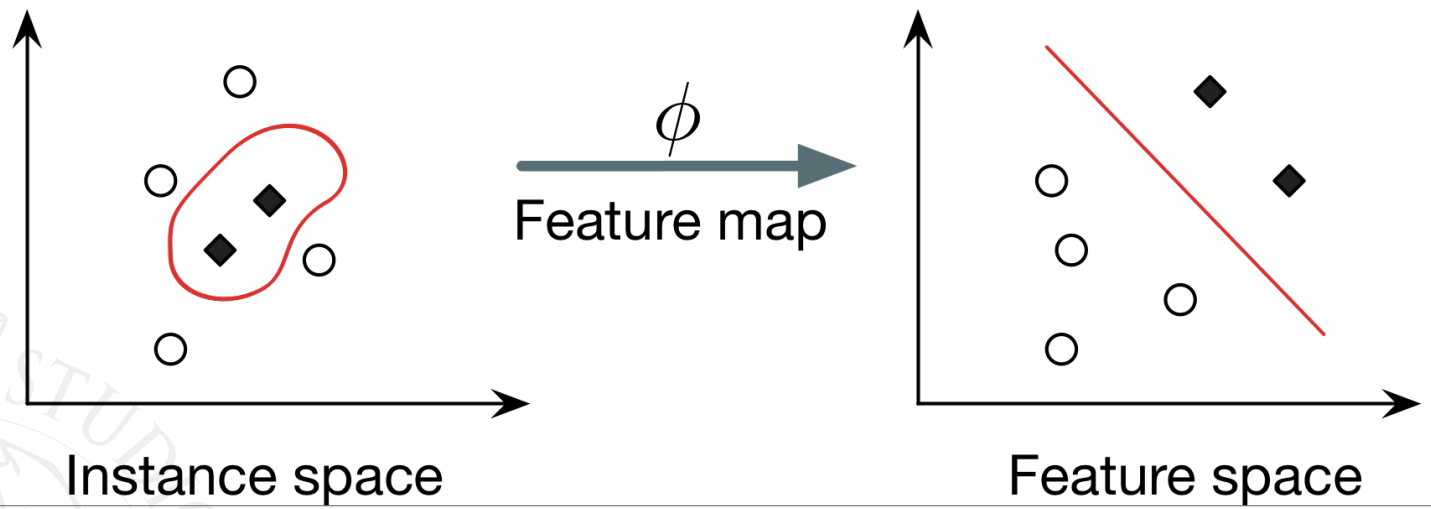
- Let $\mathcal{X}$ denote the instance space
- Let $\mathcal{F}$ denote the feature space (a Hilbert space)

- Let $\mathcal{X}$ denote the instance space
- Let $\mathcal{F}$ denote the feature space (a Hilbert space)
- A function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a valid kernel if there exists a feature map $\phi : \mathcal{X} \mapsto \mathcal{F}$ such that

$$k(x, z) = \langle \phi(x), \phi(z) \rangle \qquad \forall x, z \in \mathcal{X}$$

- Let $\mathcal{X}$ denote the instance space
- Let $\mathcal{F}$ denote the feature space (a Hilbert space)
- A function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a valid kernel if there exists a feature map $\phi : \mathcal{X} \mapsto \mathcal{F}$ such that

$$k(x, z) = \langle \phi(x), \phi(z) \rangle \qquad \forall x, z \in \mathcal{X}$$



- Let $\mathcal{X}$ denote the instance space
- Let $\mathcal{F}$ denote the feature space (a Hilbert space)
- A function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a valid kernel if there exists a feature map $\phi : \mathcal{X} \mapsto \mathcal{F}$ such that

$$k(x, z) = \langle \phi(x), \phi(z) \rangle \qquad \forall x, z \in \mathcal{X}$$

- Equivalently (Mercer's theorem): k is a valid kernel iff
  - *Symmetric*: $k(x, z) = k(z, x)$
  - *Positive semidefinite*: for every finite set of points, the matrix with entries

$$k_{ij} = k\left(x^{(i)}, x^{(j)}\right)$$

    has no negative eigenvalue

Example: support vector classification (SVC)

- Given the dataset $\mathcal{D} = \left\{ \left(x^{(i)}, y^{(i)} \right) ; i = 1, \ldots, n \right\}$

Example: support vector classification (SVC)

- Given the dataset $\mathcal{D} = \left\{ \left(x^{(i)}, y^{(i)} \right) ; i = 1, \ldots, n \right\}$
- The prediction function $f : \mathcal{X} \mapsto \mathbb{R}$ can be written as

$$f(x) = \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k \left(x^{(i)}, x \right) + b$$

- The purpose of the kernel is to measure the similarity between the test point x and every training example $x^{(i)}$

Example: support vector classification (SVC)

- Given the dataset $\mathcal{D} = \left\{ \left(x^{(i)}, y^{(i)} \right) ; i = 1, \ldots, n \right\}$
- The prediction function $f : \mathcal{X} \mapsto \mathbb{R}$ can be written as

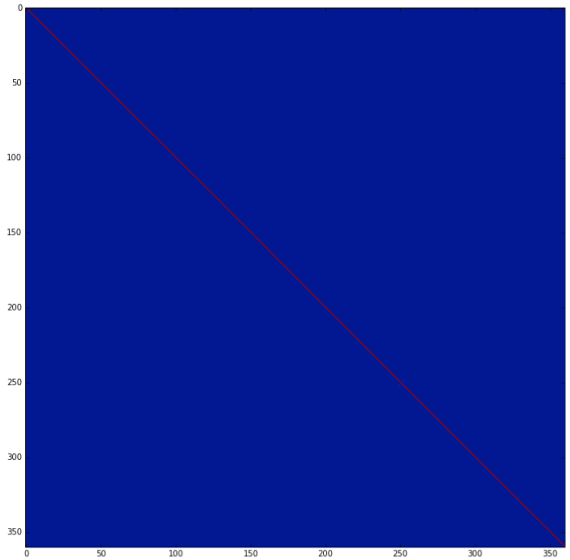
$$f(x) = \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k \left(x^{(i)}, x \right) + b$$

- The purpose of the kernel is to measure the similarity between the test point x and every training example $x^{(i)}$
- The coefficients are the solution of the QP

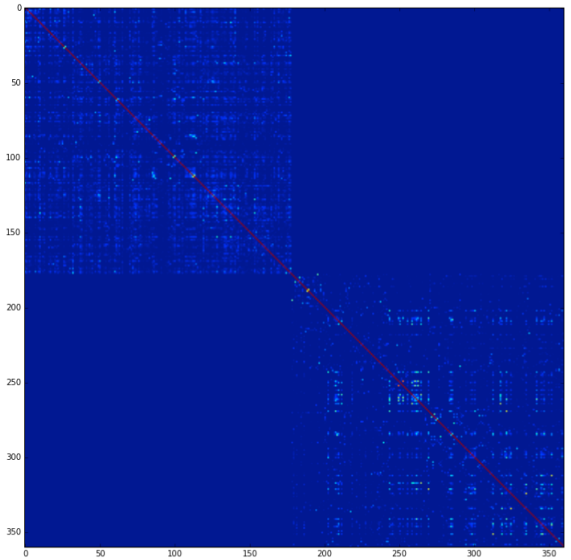
$$\begin{aligned} \min_{\alpha} \quad & \|\alpha\|_1 + \frac{1}{2} \alpha^\intercal Q \alpha \\ \text{subject to:} \quad & 0 \leq \alpha^{(i)} \leq C \qquad i = 1, \ldots, n \\ & \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} = 0 \end{aligned}$$

where $Q = (YY^\intercal) \circ K$ and K is the kernel matrix of $\mathcal{D}$

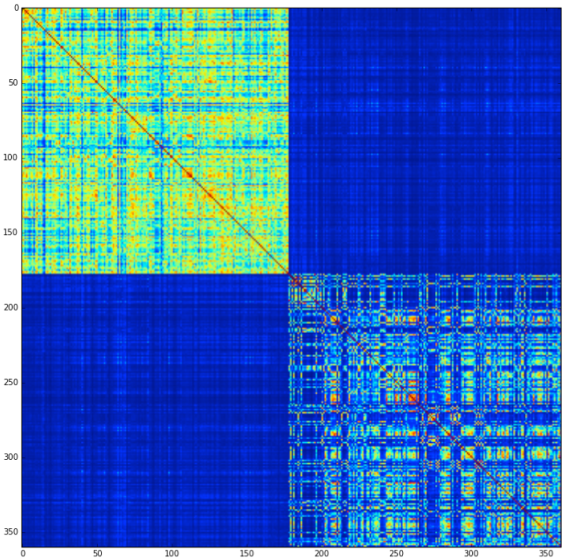
Check that your kernel looks reasonable



Diagonal



Diagonal dominant



Reasonable

Advantages of the kernel trick

- Performance: Using appropriate kernel functions we may be able to perform nonlinear classification and attain better accuracy (because of lower approximation error)

Advantages of the kernel trick

- Performance: Using appropriate kernel functions we may be able to perform nonlinear classification and attain better accuracy (because of lower approximation error)
- Efficiency (for small-medium size datasets): we may be able to compute $k(x, z)$ without computing the transformed $\phi(x)$ and $\phi(z)$ explicitly

Advantages of the kernel trick

- Performance: Using appropriate kernel functions we may be able to perform nonlinear classification and attain better accuracy (because of lower approximation error)
- Efficiency (for small-medium size datasets): we may be able to compute $k(x, z)$ without computing the transformed $\phi(x)$ and $\phi(z)$ explicitly
- Abstraction: The data type of x does not matter anymore and we can apply many existing algorithms to arbitrary objects (in particular, structured ones)

Disadvantages of kernels

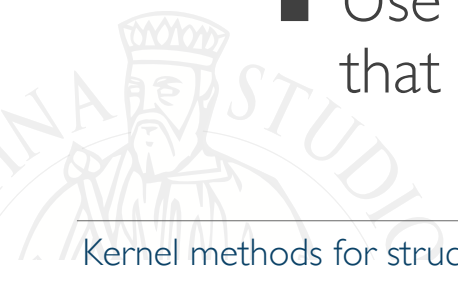
- Knowledge: Many different kernels are possible and choosing the correct one may require background knowledge that we don't have

Disadvantages of kernels

- Knowledge: Many different kernels are possible and choosing the correct one may require background knowledge that we don't have
- Efficiency: Solving a quadratic problem is prohibitive for large datasets

Disadvantages of kernels

- Knowledge: Many different kernels are possible and choosing the correct one may require background knowledge that we don't have
- Efficiency: Solving a quadratic problem is prohibitive for large datasets
- The quadratic problem may be avoided in some cases
  - Approximate the kernel matrix, e.g. using the Nyström method, or restrict to homogeneous kernels (Vedaldi et al. 2012)
  - Use kernels that correspond to very sparse feature vectors that may be obtained explicitly



Closure properties

- Suppose k_1 and k_2 are two valid (positive semi-definite) kernels with $k_j(x, z) = \langle \phi_j(x), \phi_j(z) \rangle$

- Suppose k_1 and k_2 are two valid (positive semi-definite) kernels with $k_j(x, z) = \langle \phi_j(x), \phi_j(z) \rangle$
- Then the following kernels are also valid:

$$
\begin{aligned}
k_3(x, z) &= k_1(x, z) + k_2(x, z) && \text{(sum)} \\
k_4(x, z) &= k_1(x, z) k_2(x, z) && \text{(product)} \\
k_5(x, z) &= \frac{k_1(x, z)}{\sqrt{k_1(x, x) k_1(z, z)}} && \text{(normalization)}
\end{aligned}
$$

- If k_1 and k_2 are valid kernels then there exists ϕ_1 and ϕ_2 such that

$$k_1(x, z) = \langle \phi_1(x), \phi_1(z) \rangle$$
$$k_2(x, z) = \langle \phi_2(x), \phi_2(z) \rangle$$

Sum and product of kernels

- If k_1 and k_2 are valid kernels then there exists ϕ_1 and ϕ_2 such that

$$k_1(x, z) = \langle \phi_1(x), \phi_1(z) \rangle$$
$$k_2(x, z) = \langle \phi_2(x), \phi_2(z) \rangle$$

- $k_1(x, z) + k_2(x, z) = \langle \phi_1(x) \oplus \phi_2(x), \phi_1(z) \oplus \phi_2(z) \rangle$ where $\oplus$ denotes the concatenation operator

- If k_1 and k_2 are valid kernels then there exists ϕ_1 and ϕ_2 such that

$$k_1(x, z) = \langle \phi_1(x), \phi_1(z) \rangle$$
$$k_2(x, z) = \langle \phi_2(x), \phi_2(z) \rangle$$

- $k_1(x, z) + k_2(x, z) = \langle \phi_1(x) \oplus \phi_2(x), \phi_1(z) \oplus \phi_2(z) \rangle$ where $\oplus$ denotes the concatenation operator

- $k_1(x, z)k_2(x, z) = \langle \phi_1(x) \otimes \phi_2(x), \phi_1(z) \otimes \phi_2(z) \rangle$ where $\otimes$ denotes the Kronecker product

$$a \otimes b = [a_1 b, a_2 b, \cdots, a_p b]$$

- Suppose k is a valid kernel; then

$$\tilde{k}(x, z) \doteq \frac{k(x, z)}{\sqrt{k(x, x)k(z, z)}}$$

  is also a valid kernel

- Suppose k is a valid kernel; then

$$\tilde{k}(x, z) \doteq \frac{k(x, z)}{\sqrt{k(x, x)k(z, z)}}$$

  is also a valid kernel

- Easy to prove: if $k(x, z) = \langle \phi(x), \phi(z) \rangle$ define

$$\tilde{\phi}(x) = \frac{\phi(x)}{\|\phi(x)\|}$$

  and check that

$$\tilde{k}(x, z) = \langle \tilde{\phi}(x), \tilde{\phi}(z) \rangle$$

Tensor product and direct sum kernels

- Suppose $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$

Tensor product and direct sum kernels

- Suppose $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$
- If κ_d, for $d = 1, \ldots, 2$ are valid kernels on $\mathcal{X}_d \times \mathcal{X}_d$, it is easy to construct valid kernels on tuples, e.g.,
  - Tensor product kernel

$$(\kappa_1 \otimes \kappa_2)((x_1, x_2), (x_2, x_2)) \doteq \kappa_1(x_1, z_1)\kappa_2(x_2, z_2)$$

  - Direct sum kernel

$$(\kappa_1 \oplus \kappa_2)((x_1, x_2), (x_2, x_2)) \doteq \kappa_1(x_1, z_1) + \kappa_2(x_2, z_2)$$

Convolution (or decomposition) kernels (Haussler 1999)

- Basic idea: decompose a composite instance space $\mathcal{X}$ into spaces representing "parts" $\mathcal{X}_1, \ldots, \mathcal{X}_D$

Convolution (or decomposition) kernels (Haussler 1999)

- Basic idea: decompose a composite instance space $\mathcal{X}$ into spaces representing "parts" $\mathcal{X}_1, \ldots, \mathcal{X}_D$
- Introduce a decomposition relation

$$R \subset \mathcal{X}_1 \times \ldots \times \mathcal{X}_D \times \mathcal{X}$$

Convolution (or decomposition) kernels (Haussler 1999)

- Basic idea: decompose a composite instance space $\mathcal{X}$ into spaces representing "parts" $\mathcal{X}_1, \ldots, \mathcal{X}_D$
- Introduce a decomposition relation

$$R \subset \mathcal{X}_1 \times \ldots \times \mathcal{X}_D \times \mathcal{X}$$

- For all $x_d \in \mathcal{X}_d, d = 1, \ldots, D$ and $x \in \mathcal{X}$, $(x_1, \ldots, x_D, x) \in R$ iff $x_1, \ldots, x_D$ are the parts of x

Convolution (or decomposition) kernels (Haussler 1999)

- Basic idea: decompose a composite instance space $\mathcal{X}$ into spaces representing "parts" $\mathcal{X}_1, \ldots, \mathcal{X}_D$
- Introduce a decomposition relation

$$R \subset \mathcal{X}_1 \times \ldots \times \mathcal{X}_D \times \mathcal{X}$$

- For all $x_d \in \mathcal{X}_d, d = 1, \ldots, D$ and $x \in \mathcal{X}$, $(x_1, \ldots, x_D, x) \in R$ iff $x_1, \ldots, x_D$ are the parts of x
- Notation:

$$R^{-1}(x) \doteq \{(x_1, \ldots, x_D) : (x_1, \ldots, x_D, x) \in R\}$$

Example on strings

- Let $\mathcal{X} = \Sigma^*$ (the set of all strings over a finite alphabet Σ)

Example on strings

- Let $\mathcal{X} = \Sigma^*$ (the set of all strings over a finite alphabet Σ)
- Let $D = 2$ and $\mathcal{X}_1 = \mathcal{X}_2 = \mathcal{X}$

Example on strings

- Let $\mathcal{X} = \Sigma^*$ (the set of all strings over a finite alphabet Σ)
- Let $D = 2$ and $\mathcal{X}_1 = \mathcal{X}_2 = \mathcal{X}$
- Define R so that $(x_1, x_2, x) \in R$ iff x_1 is a prefix of x and x_2 the complementary suffix

$$(\texttt{TATAG}, \texttt{ACGA}, \texttt{TATAGACGA}) \in R$$

$$(\texttt{TAT}, \texttt{ACGA}, \texttt{TATAGACGA}) \notin R$$

Example on trees

- Let $\mathcal{X}$ be the set of all parse tree over nonterminals $\mathcal{N}$ (terminal symbols omitted)

Example on trees

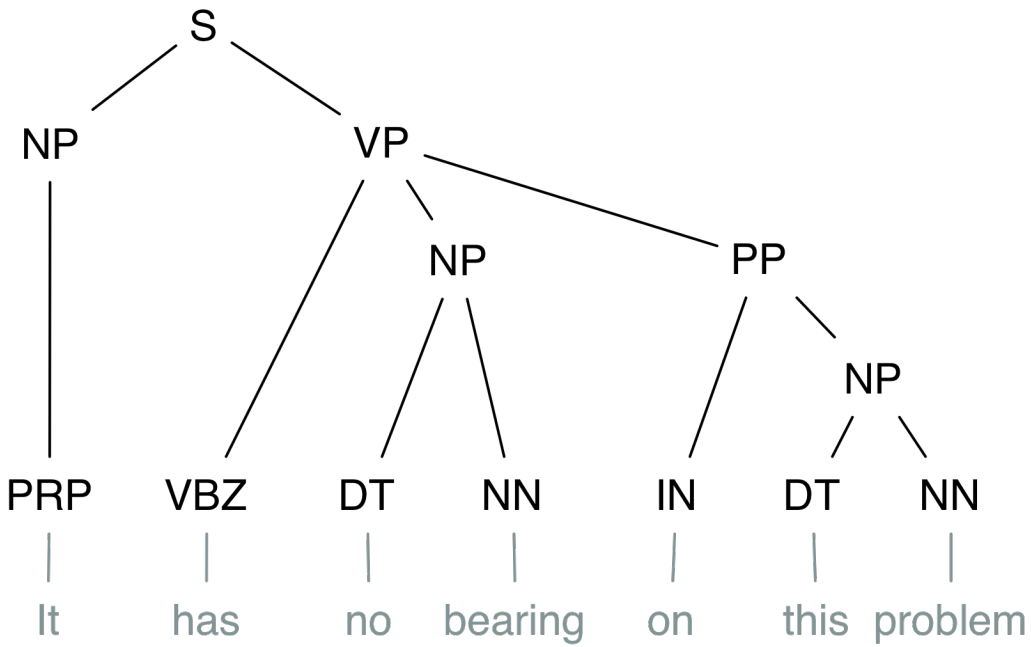
- Let $\mathcal{X}$ be the set of all parse tree over nonterminals $\mathcal{N}$ (terminal symbols omitted)
- A co-rooted subtree of x is a tree obtained as follow:
  - Take a complete subtree x' of x
  - Remove some complete subtrees from x'
  - Replace the roots of the removed subtrees

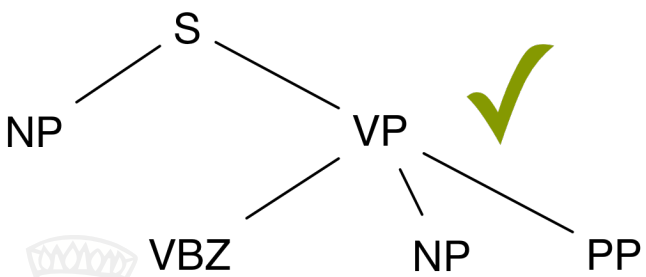
Example on trees

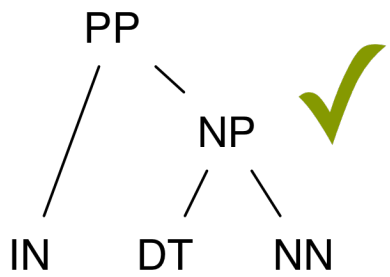
- Let $\mathcal{X}$ be the set of all parse tree over nonterminals $\mathcal{N}$ (terminal symbols omitted)
- A co-rooted subtree of x is a tree obtained as follow:
  - Take a complete subtree x' of x
  - Remove some complete subtrees from x'
  - Replace the roots of the removed subtrees
- A co-rooted substree never splits across a production rule

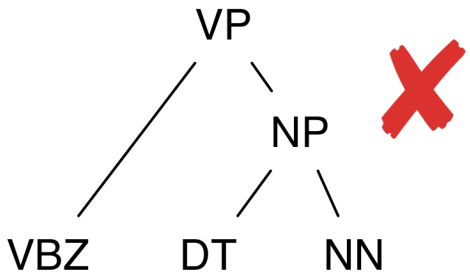
$$(t, x) \in R \quad \text{iff } t \text{ is a co-rooted subtree of } x$$

Co-rooted subtrees (Collins et al. 2001)









Example on graphs

- Graph: $G = (V, E)$
- Let $\mathcal{X}$ be the set of all labeled undirected graphs

Example on graphs

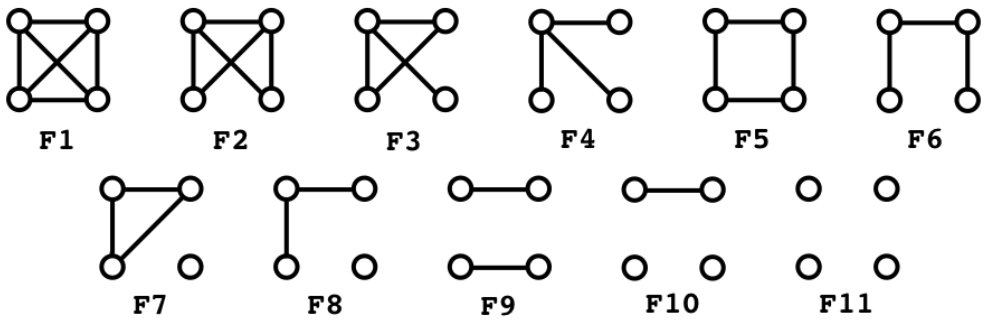
- Graph: $G = (V, E)$
- Let $\mathcal{X}$ be the set of all labeled undirected graphs
- A path (or walk) π is a sequence of vertices $\pi_1, \ldots, \pi_{|\pi|}$ such that $\pi_j \in V$ and $(\pi_j, \pi_{j+1}) \in E$

Example on graphs

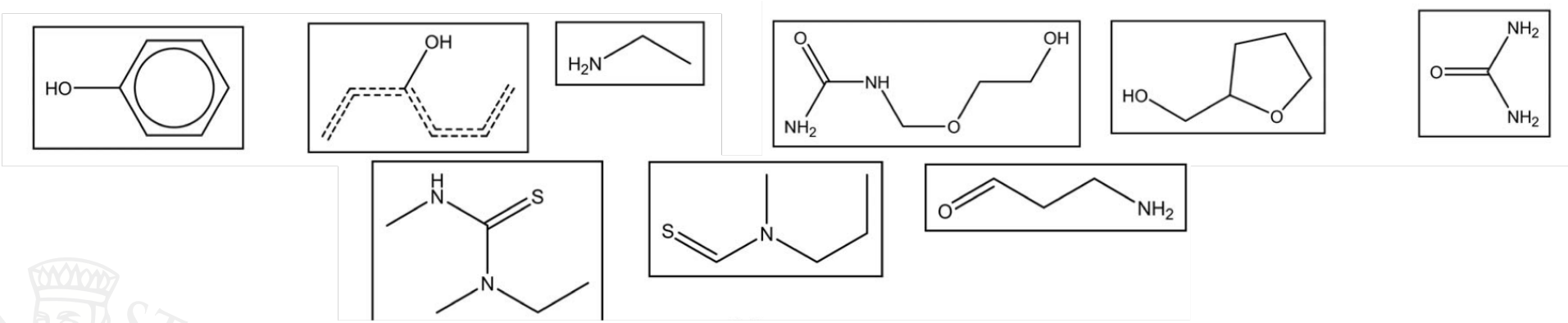
- Graph: $G = (V, E)$
- Let $\mathcal{X}$ be the set of all labeled undirected graphs
- A path (or walk) π is a sequence of vertices $\pi_1, \ldots, \pi_{|\pi|}$ such that $\pi_j \in V$ and $(\pi_j, \pi_{j+1}) \in E$
- If $\ell : V \mapsto \mathcal{L}$ is the vertex labeling function, the sequence $\ell(\pi_1), \ldots, \ell(\pi_{|\pi|})$ is a labeled path of x

$$(\pi, x) \in R \quad \text{iff } \pi \text{ is a labeled path of } x$$

- Simply: $(g, x) \in R$ iff g is a subgraph of x



All graphlets of 4 nodes (Shervashidze et al. 2009; Yanardag et al. 2015)



Frequent subgraph mining (Deshpande et al. 2005; Wale et al. 2008)

Convolution kernels

- Assume the instance space $\mathcal{X}$ can be decomposed into subspaces via a decomposition relation $R \subset \mathcal{X}_1 \times \ldots \times \mathcal{X}_D \times \mathcal{X}$

- Assume the instance space $\mathcal{X}$ can be decomposed into subspaces via a decomposition relation $R \subset \mathcal{X}_1 \times \ldots \times \mathcal{X}_D \times \mathcal{X}$
- Suppose we have a valid kernel κ_d over all subspaces $\mathcal{X}_d$ for $d = 1, \ldots, D$

- Assume the instance space $\mathcal{X}$ can be decomposed into subspaces via a decomposition relation $R \subset \mathcal{X}_1 \times \ldots \times \mathcal{X}_D \times \mathcal{X}$
- Suppose we have a valid kernel κ_d over all subspaces $\mathcal{X}_d$ for $d = 1, \ldots, D$
- Then the following kernel is valid over $\mathcal{X}$ (Haussler 1999):

$$K(x, z) = \sum_{\substack{(x_1, \ldots, x_d) \in R^{-1}(x) \\ (z_1, \ldots, z_d) \in R^{-1}(z)}} \prod_{d=1}^{D} \kappa_d(x_d, z_d)$$

Part 3

Graph kernels

Path kernels

- Graph: $G = (V, E)$

Path kernels

- Graph: $G = (V, E)$
- Labels may be attached to vertices and/or edges

Path kernels

- Graph: $G = (V, E)$
- Labels may be attached to vertices and/or edges
- Let κ_{node} and κ_{edge} be valid kernels on node labels and edge labels, respectively

Path kernels

- Graph: $G = (V, E)$
- Labels may be attached to vertices and/or edges
- Let κ_{node} and κ_{edge} be valid kernels on node labels and edge labels, respectively
- Define the path kernel as the *tensor product kernel*

$$\kappa_{\text{path}}(\pi, \pi') = \mathbb{1}\{|\pi| = |\pi'|\} \cdot \prod_{j=1}^{|\pi|} \kappa_{\text{node}}\left(\ell(\pi_j), \ell(\pi'_j)\right) \cdot \prod_{j=1}^{|\pi|-1} \kappa_{\text{edge}}\left(\ell((\pi_j, \pi_{j+1})), \ell((\pi'_j, \pi'_{j+1}))\right)$$

From path kernels to graph kernels

- Random walk kernels count the number of matching walks:

$$K(G, G') = \sum_{\pi \in R^{-1}(G)} \sum_{\pi' \in R^{-1}(G')} \kappa_{\text{path}}(\pi, \pi')$$

where $(\pi, G) \in R$ iff π is a path in G

From path kernels to graph kernels

- Random walk kernels count the number of matching walks:

$$K(G, G') = \sum_{\pi \in R^{-1}(G)} \sum_{\pi' \in R^{-1}(G')} \kappa_{\text{path}}(\pi, \pi')$$

  where $(\pi, G) \in R$ iff π is a path in G

- Computing $\phi(G)$ is NP-complete (Gärtner et al. 2003) — proof by reduction to finding a Hamiltonian path

From path kernels to graph kernels

- Random walk kernels count the number of matching walks:

$$K(G, G') = \sum_{\pi \in R^{-1}(G)} \sum_{\pi' \in R^{-1}(G')} \kappa_{\text{path}}(\pi, \pi')$$

  where $(\pi, G) \in R$ iff π is a path in G

- Computing $\phi(G)$ is NP-complete (Gärtner et al. 2003) — proof by reduction to finding a Hamiltonian path

- Many possible approaches, we will briefly review the following ideas:
  - Use a marginalized kernel (Kashima et al. 2003)
  - Use product graphs (Gärtner et al. 2003)
  - Use shortest-paths (Borgwardt et al. 2005a)

Marginalized graph kernels

- Let $\mathbb{P}(\pi|G)$ denote the probability of a random walk π in G

Marginalized graph kernels

- Let $\mathbb{P}(\pi|G)$ denote the probability of a random walk π in G
- To compute it:
  - Sample the first node π_1 from a start distribution p_s
  - At the j-th step, sample the next node π_j from a transition distribution $p_t(\pi_j|\pi_{j-1})$
  - Allow termination using a stop distribution p_q such that

$$\sum_{v \in V} p_t(v|w) + p_q(w) = 1 \quad \forall w \in V$$

- Let $\mathbb{P}(\pi|G)$ denote the probability of a random walk π in G
- To compute it:
  - Sample the first node π_1 from a start distribution p_s
  - At the j-th step, sample the next node π_j from a transition distribution $p_t(\pi_j|\pi_{j-1})$
  - Allow termination using a stop distribution p_q such that

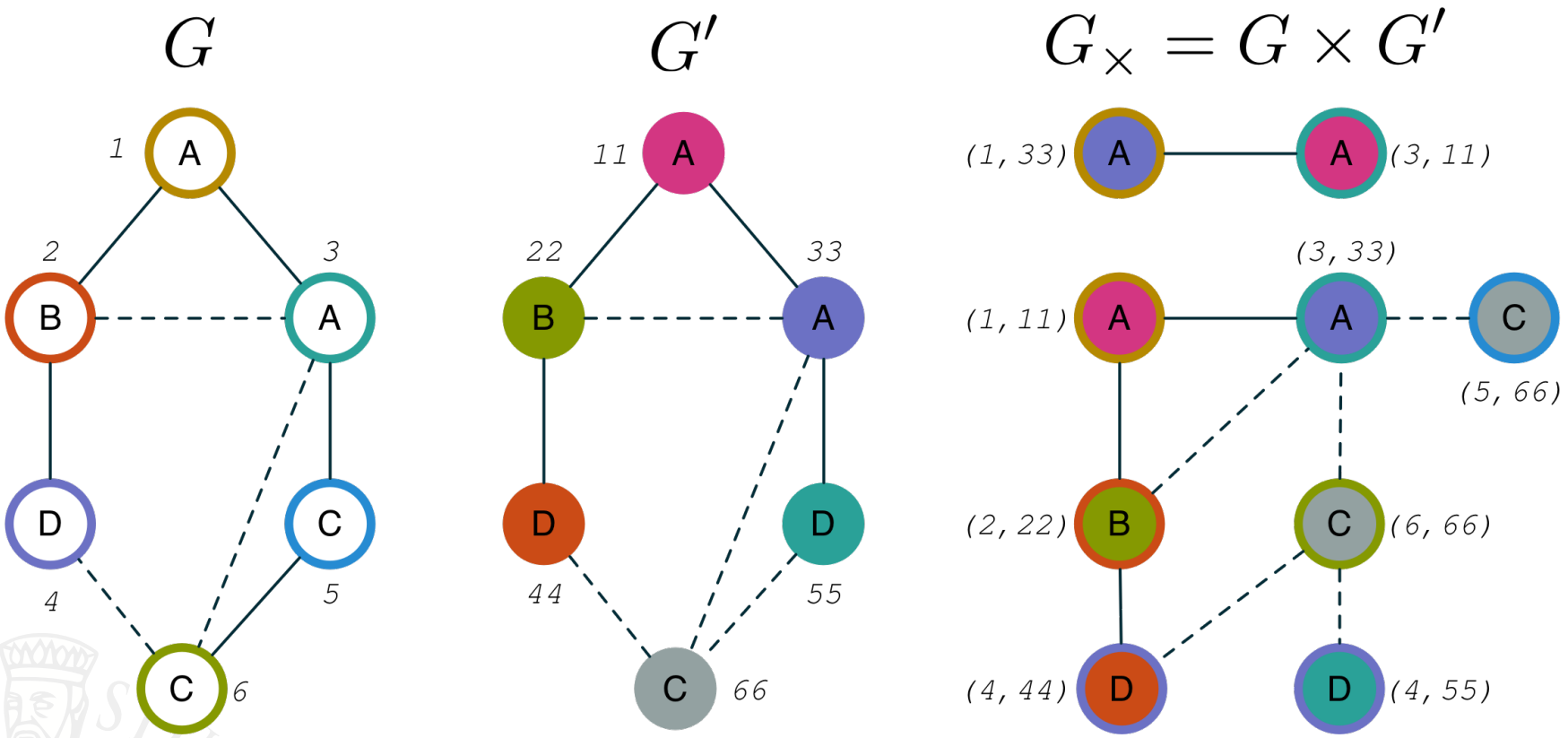
$$\sum_{v \in V} p_t(v|w) + p_q(w) = 1 \quad \forall w \in V$$

- Define the graph kernel as (Kashima et al. 2003)

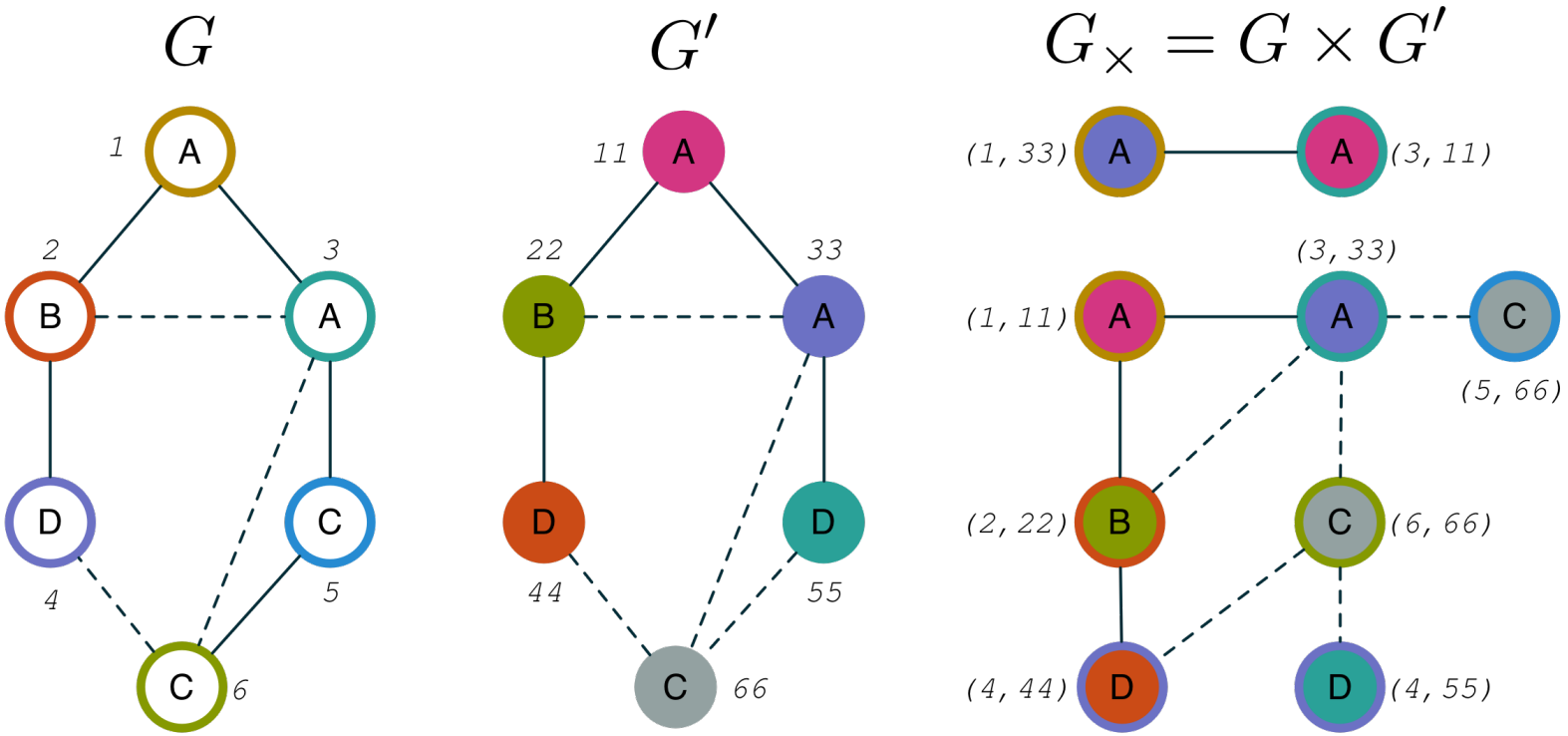
$$k(G, G') = \sum_{\pi} \sum_{\pi'} k_p(\pi, \pi') \, \mathbb{P}(\pi|G) \, \mathbb{P}(\pi'|G')$$

- Given two graphs G and G' they direct product is the graph $G \times G'$

$$V_\times = \{(v, v') \in V \times V' : \ell(v) = \ell(v')\}$$
$$E_\times = \{((u, u'), (v, v')) \in V_\times^2 : (u, v) \in E, (u', v') \in E', \ell(u, v) = \ell(u', v')\}$$

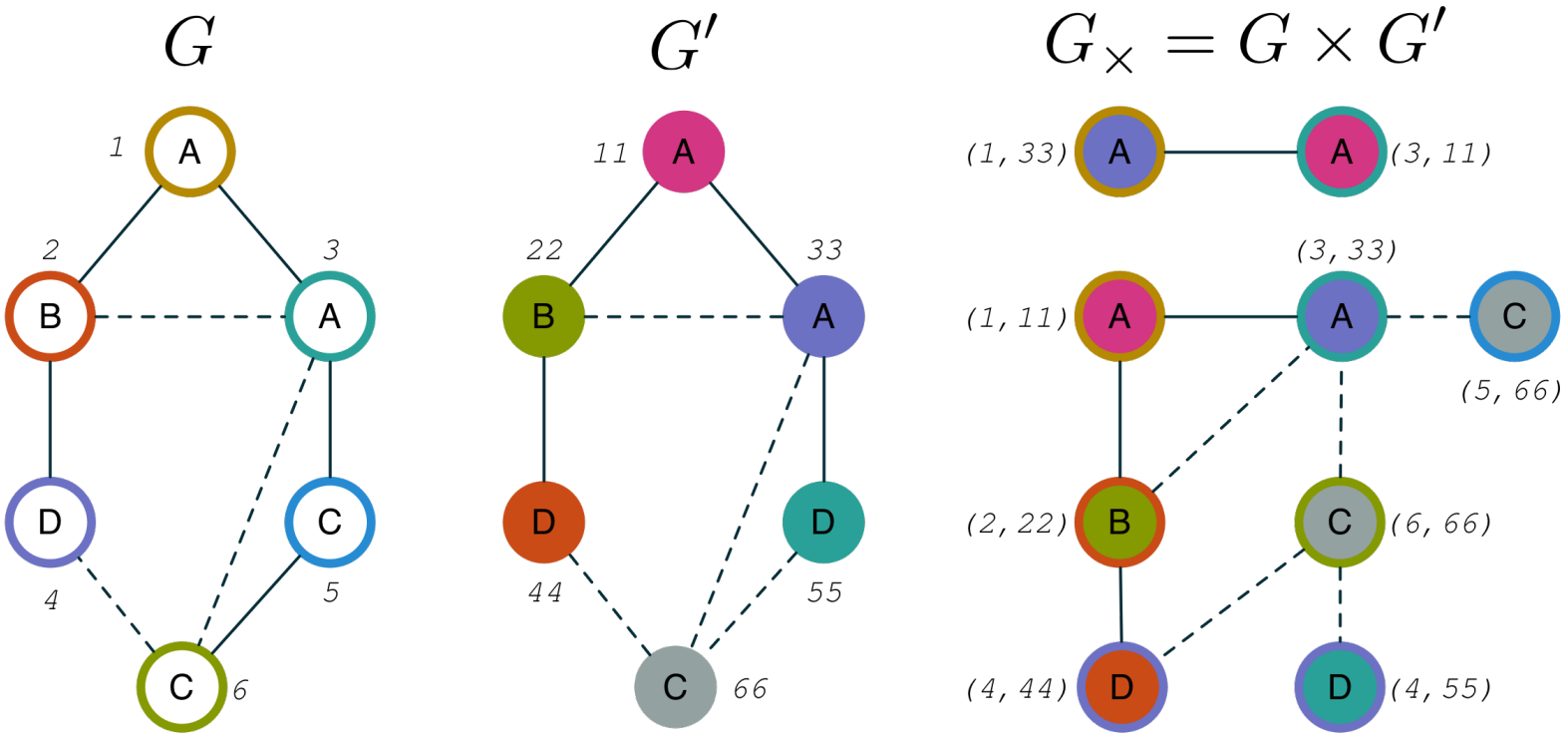


Product graphs



- Main result: There is a bijection between any walk in $G_\times$ and a common walk in G and G' (Gärtner et al. 2003)

Product graphs



- Main result: There is a bijection between any walk in $G_\times$ and a common walk in G and G' (Gärtner et al. 2003)

- Hence the kernel can be computed as

$$K(G, G') = \sum_{i,j \in V_\times} \left[\sum_{n=0}^{\infty} \lambda_n E_\times^n \right]_{ij}$$

- Need to compute

$$\sum_{n=0}^{\infty} \lambda_n E_{\times}^n$$

- Need to compute

$$\sum_{n=0}^{\infty} \lambda_n E_{\times}^n$$

- The series converges when λ_n are properly choosen

- Need to compute

$$\sum_{n=0}^{\infty} \lambda_n E_{\times}^n$$

- The series converges when λ_n are properly choosen
- One option is the geometric series: $\lambda_n = \gamma^n$ for $\gamma < 1$:

$$\lim_{n \to \infty} \sum_{i=0}^{n} \gamma^i = \frac{1}{1-\gamma}$$

- Need to compute

$$\sum_{n=0}^{\infty} \lambda_n E_{\times}^n$$

- The series converges when λ_n are properly choosen
- One option is the geometric series: $\lambda_n = \gamma^n$ for $\gamma < 1$:

$$\lim_{n\to\infty} \sum_{i=0}^{n} \gamma^i = \frac{1}{1-\gamma}$$

- Note that any element of E^i is bounded by d^i where d is the maximum degree in G — hence choose $\gamma < 1/d$ obtaining

$$\lim_{n\to\infty} \sum_{i=0}^{n} \gamma^i E_{\times}^i = (I - \gamma E_{\times})^{-1}$$

Tottering

- Main problem of random walk kernels

Tottering

- Main problem of random walk kernels
- Walks allow repetitions of nodes, hence small identical subgraphs can lead to artificially high values of the kernel

Tottering

- Main problem of random walk kernels
- Walks allow repetitions of nodes, hence small identical subgraphs can lead to artificially high values of the kernel
- One alternative is to use cycles (Horváth et al. 2004)

Tottering

- Main problem of random walk kernels
- Walks allow repetitions of nodes, hence small identical subgraphs can lead to artificially high values of the kernel
- One alternative is to use cycles (Horváth et al. 2004)
- Another alternative is to use shortest-paths (Borgwardt et al. 2005a)

Shortest-path graph kernel

- Floyd-Warshall transformation: transform $G = (V, E)$ into $S = (V, \mathcal{E})$ where
  - $(u, v) \in \mathcal{E}$ iff u and v are mutually reachable
  - $\sigma(u, v)$ is the (labeled) shortest path between u and v in G

Shortest-path graph kernel

- Floyd-Warshall transformation: transform $G = (V, E)$ into $S = (V, \mathcal{E})$ where
  - $(u, v) \in \mathcal{E}$ iff u and v are mutually reachable
  - $\sigma(u, v)$ is the (labeled) shortest path between u and v in G

- Define the shortest-path kernel as

$$K_{\text{sp}}(G, G') = \sum_{e \in \mathcal{E}} \sum_{e' \in \mathcal{E}'} \kappa_{\text{path}}(\sigma(e), \sigma(e'))$$

Shortest-path graph kernel

- Floyd-Warshall transformation: transform $G = (V, E)$ into $S = (V, \mathcal{E})$ where
  - $(u, v) \in \mathcal{E}$ iff u and v are mutually reachable
  - $\sigma(u, v)$ is the (labeled) shortest path between u and v in G
- Define the shortest-path kernel as

$$K_{\text{sp}}(G, G') = \sum_{e \in \mathcal{E}} \sum_{e' \in \mathcal{E}'} \kappa_{\text{path}}(\sigma(e), \sigma(e'))$$

- Positive definite since it is a special case of a convolution kernel

Shortest-path graph kernel

- Floyd-Warshall transformation: transform $G = (V, E)$ into $S = (V, \mathcal{E})$ where
  - $(u, v) \in \mathcal{E}$ iff u and v are mutually reachable
  - $\sigma(u, v)$ is the (labeled) shortest path between u and v in G

- Define the shortest-path kernel as

$$K_{\text{sp}}(G, G') = \sum_{e \in \mathcal{E}} \sum_{e' \in \mathcal{E}'} \kappa_{\text{path}}(\sigma(e), \sigma(e'))$$

- Positive definite since it is a special case of a convolution kernel

- Running time is dominated by the pairwise comparisons, that take $O(V^4)$ — Floyd-Warshall runs in $O(V^3)$

- Two graphs G and G' are isomorphic (written $G \approx G'$) if there exists a bijection $f : V \mapsto V'$ (called an isomorphism) such that $\{u, v\} \in E$ iff $\{f(u), f(v)\} \in E'$

- Two graphs G and G' are isomorphic (written $G \approx G'$) if there exists a bijection $f : V \mapsto V'$ (called an isomorphism) such that $\{u, v\} \in E$ iff $\{f(u), f(v)\} \in E'$
- Very recently found to have quasi-polynomial complexity (Babai 2015)

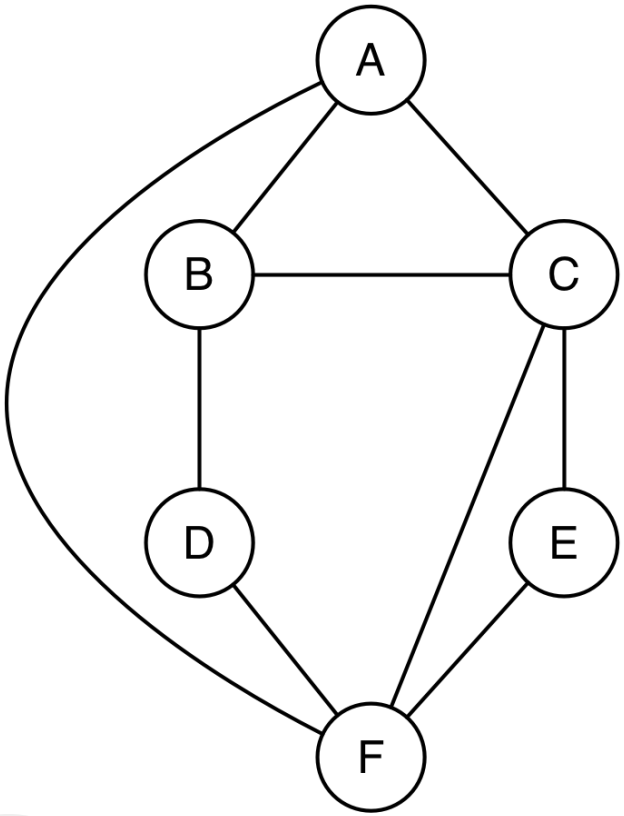
Graph isomorphism

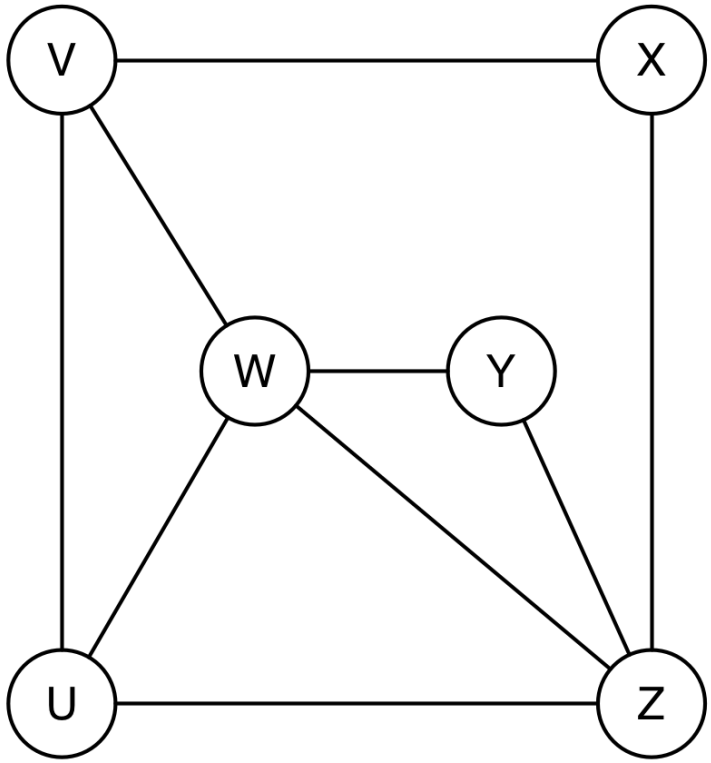
- Two graphs G and G' are isomorphic (written $G \approx G'$) if there exists a bijection $f : V \mapsto V'$ (called an isomorphism) such that $\{u, v\} \in E$ iff $\{f(u), f(v)\} \in E'$
- Very recently found to have quasi-polynomial complexity (Babai 2015)
- Still, practical algorithms employ different strategies, e.g. based on vertex recoloring via propagation mechanisms

Graph isomorphism

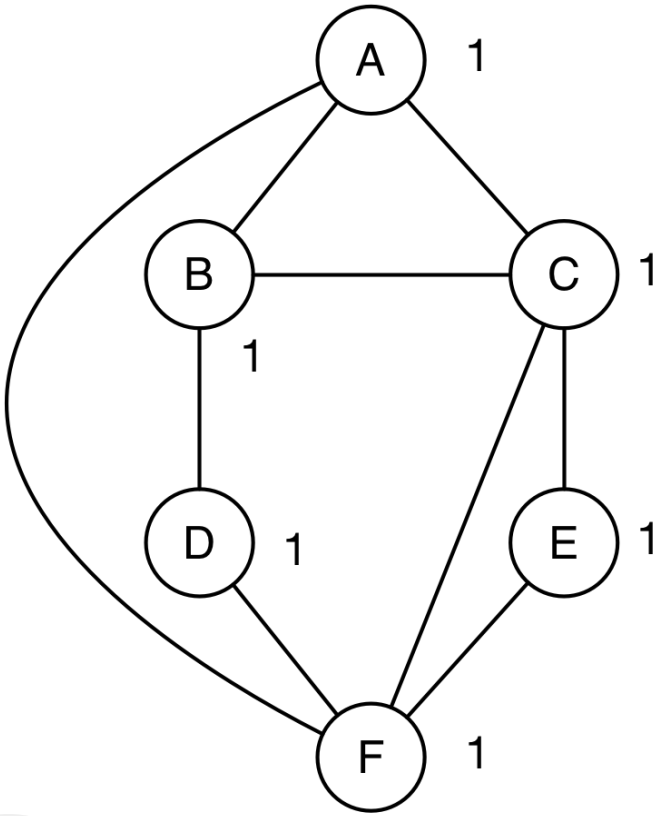
- Two graphs G and G' are isomorphic (written $G \approx G'$) if there exists a bijection $f : V \mapsto V'$ (called an isomorphism) such that $\{u, v\} \in E$ iff $\{f(u), f(v)\} \in E'$
- Very recently found to have quasi-polynomial complexity (Babai 2015)
- Still, practical algorithms employ different strategies, e.g. based on vertex recoloring via propagation mechanisms
- The 1-dimensional Weisfeiler-Lehman test is suitable for deriving a kernel (Shervashidze et al. 2011)

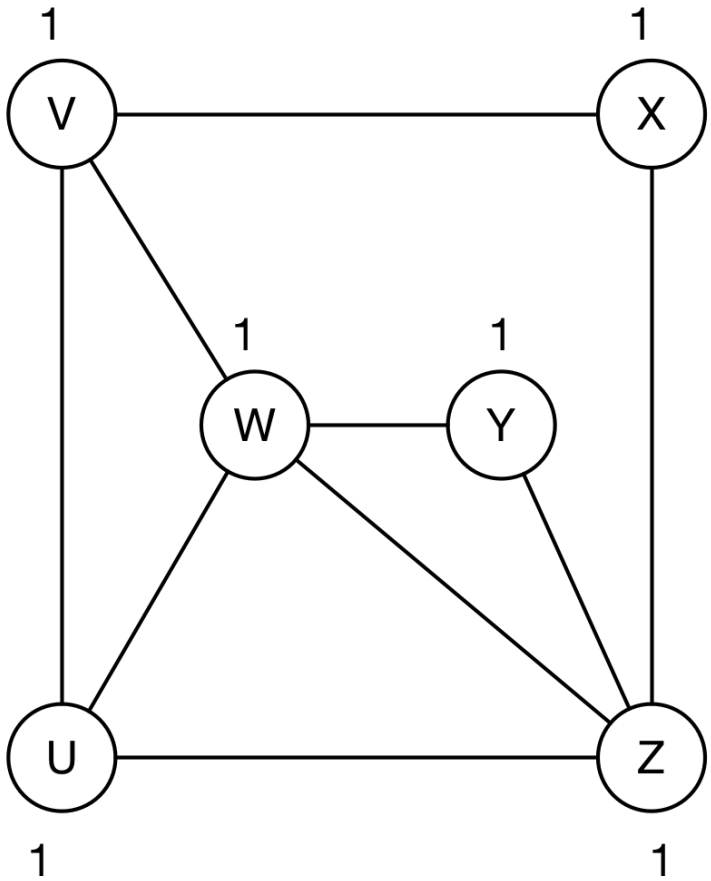
Are these two graphs isomorphic?



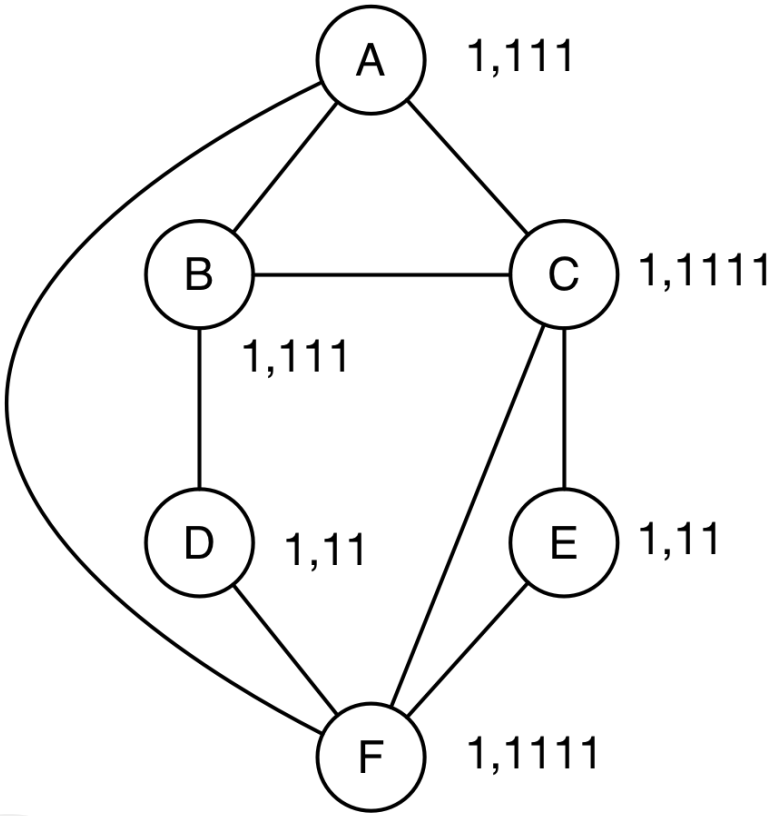


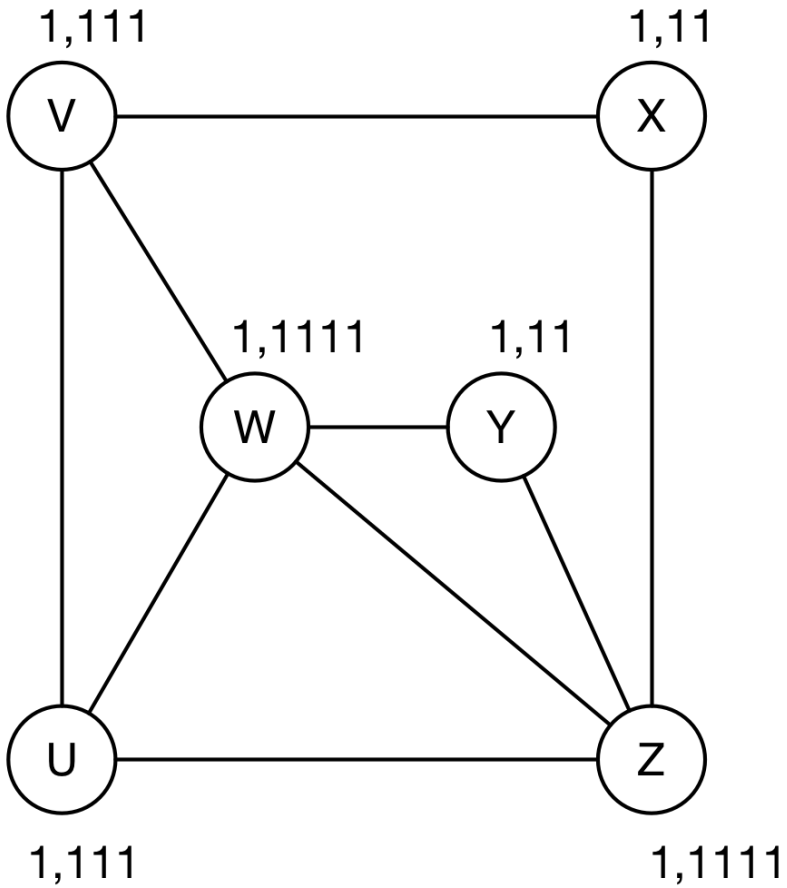
W-L test: start with all ones



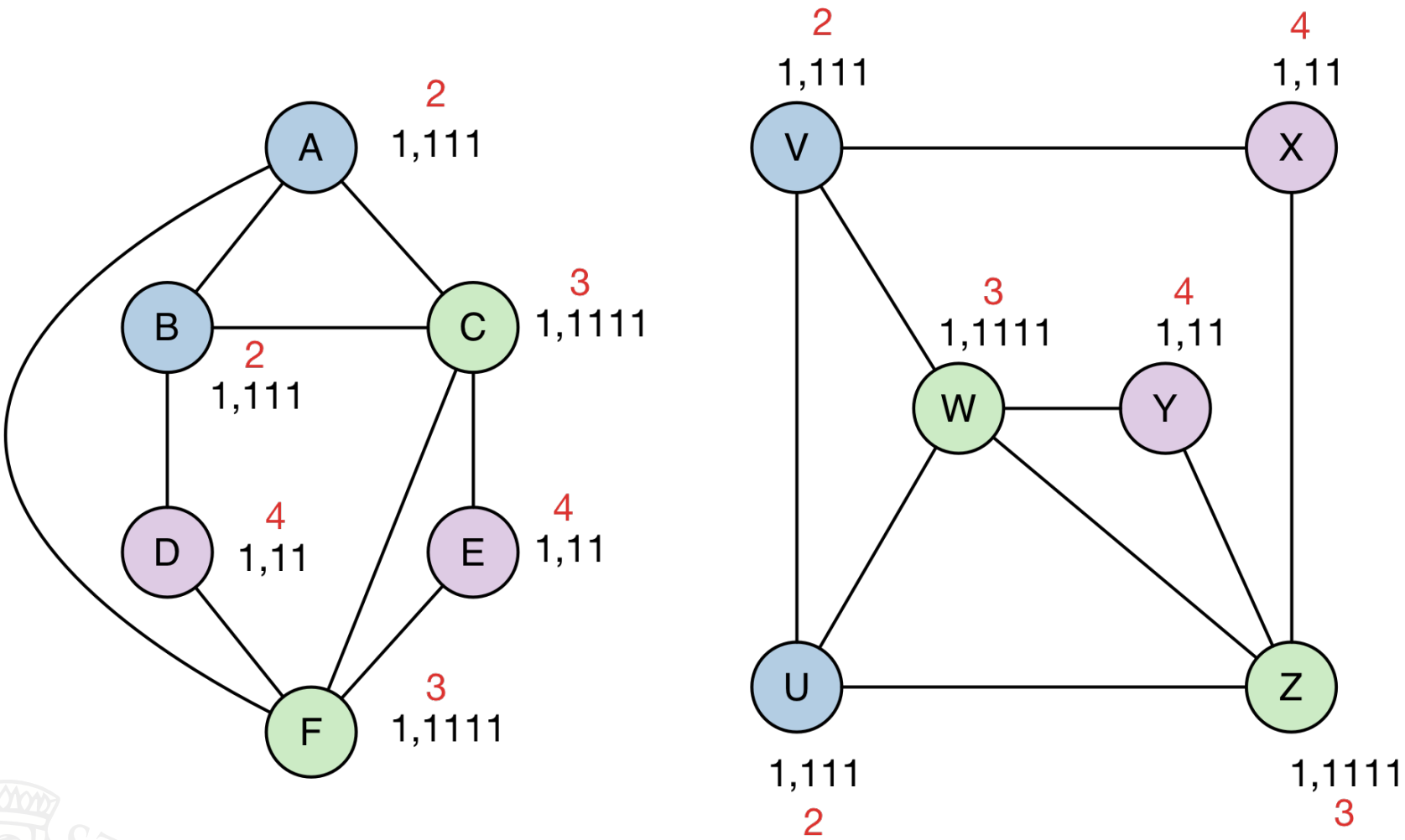


W-L test: propagate

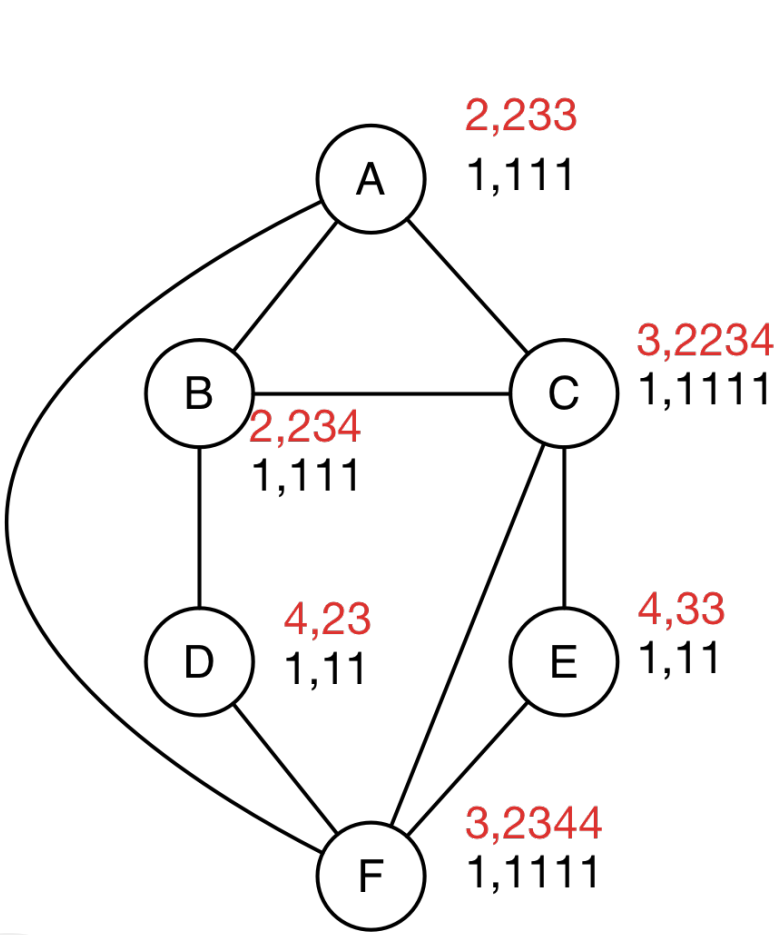


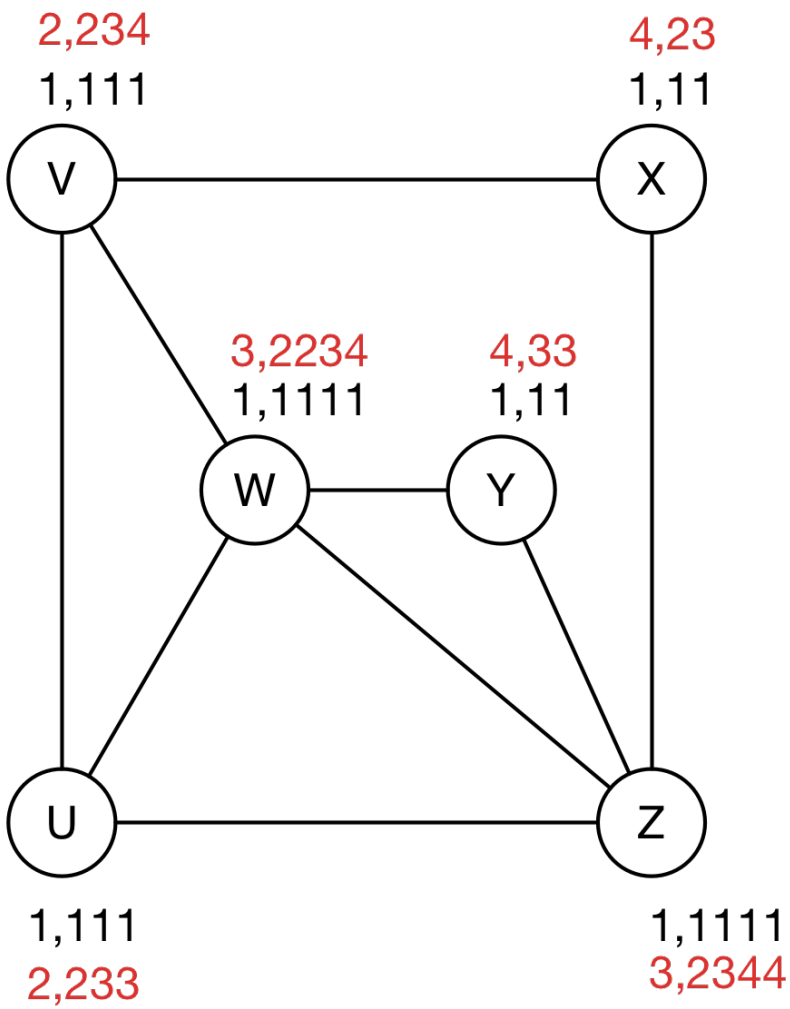


W-L test: recolor

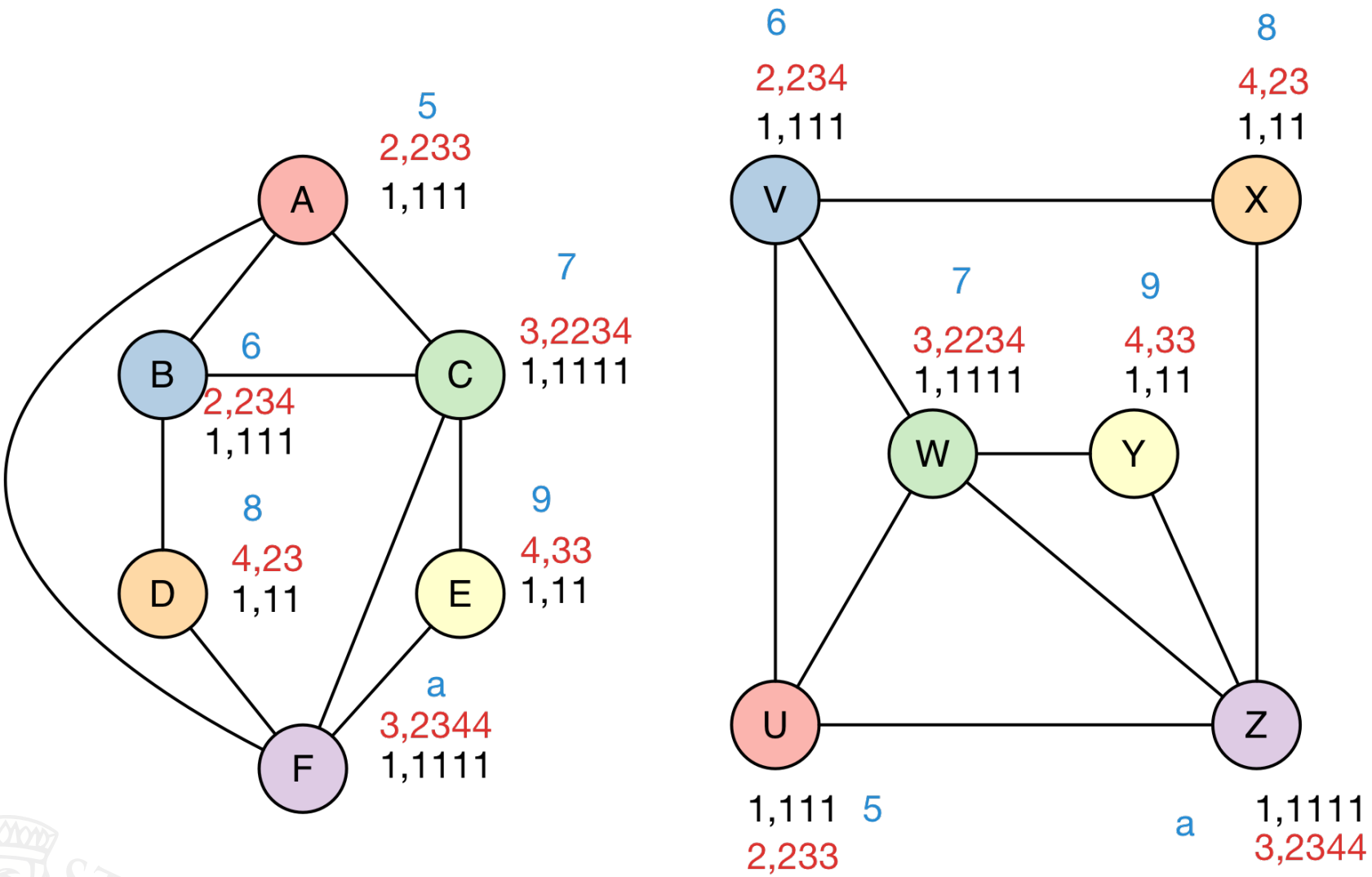


W-L test: propagate again

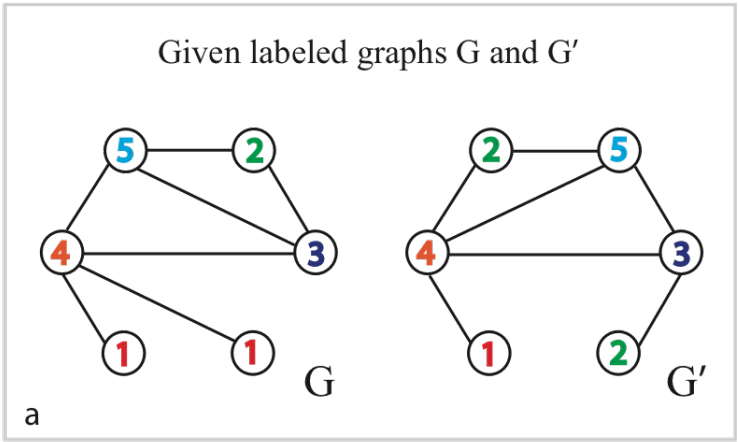


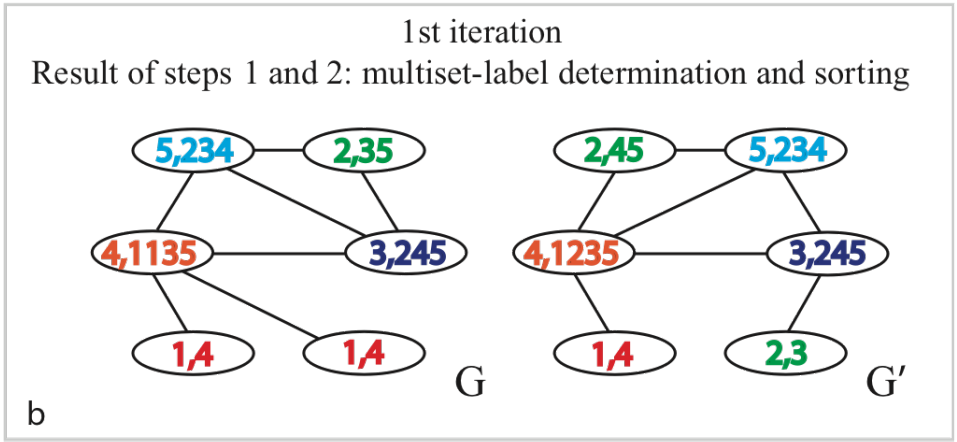


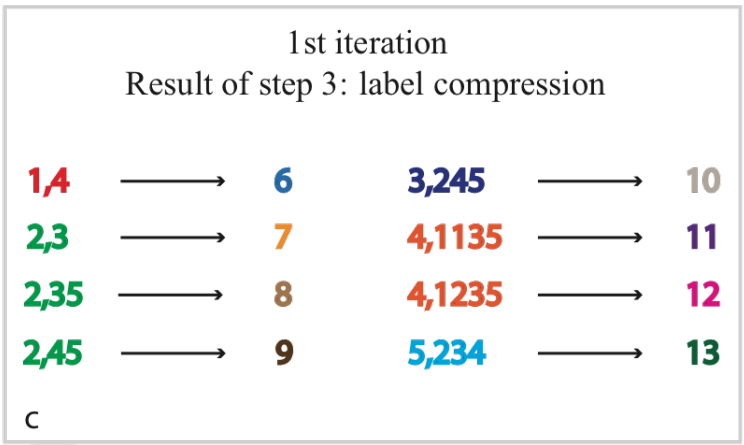
W-L test: recolor again: isomorphic!

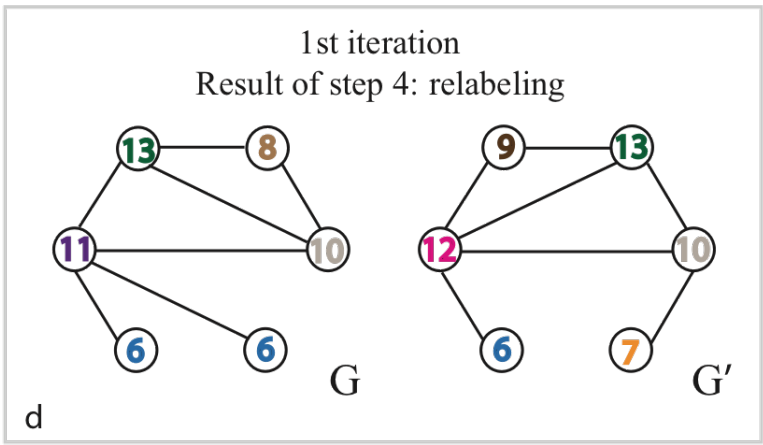


W-L Graph kernel: propagation

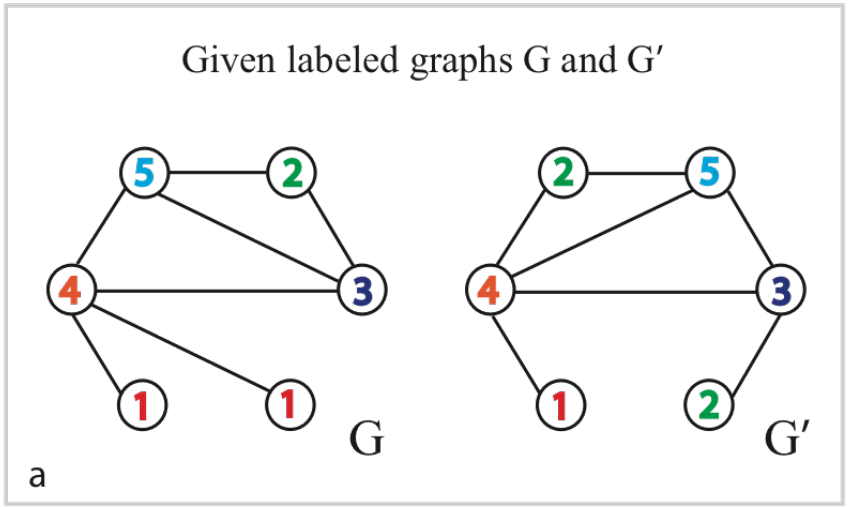


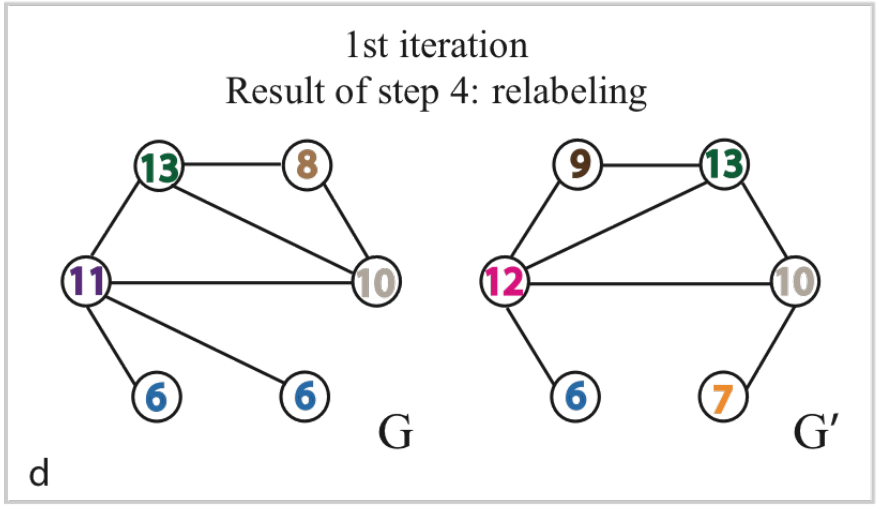






W-L Graph kernel: feature vectors (Shervashidze et al. 2011)





End of the 1st iteration
Feature vector representations of G and G′

$$\varphi^{(1)}_{WLsubtree}(\mathrm{G}) = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$$

$$\varphi^{(1)}_{WLsubtree}(\mathrm{G'}) = (\underbrace{1, 2, 1, 1, 1}_{\text{Counts of original node labels}}, \underbrace{1, 1, 0, 1, 1, 0, 1, 1}_{\text{Counts of compressed node labels}})$$

$$k^{(1)}_{WLsubtree}(\mathrm{G},\mathrm{G}') = \langle \varphi^{(1)}_{WLsubtree}(\mathrm{G}), \varphi^{(1)}_{WLsubtree}(\mathrm{G}') \rangle = 11.$$

W-L Graph kernel: Remarks

- Concatenating feature vectors is the same as summing kernels so effectively we have mapped graphs into a sequence of graphs $G_0 = G, G_1, \ldots, G_T$ and computed

$$K_{\text{WL}}(G, G') = \sum_{t=0}^{T} k(G_t, G'_t)$$

W-L Graph kernel: Remarks

- Concatenating feature vectors is the same as summing kernels so effectively we have mapped graphs into a sequence of graphs $G_0 = G, G_1, \ldots, G_T$ and computed

$$K_{\text{WL}}(G, G') = \sum_{t=0}^{T} k(G_t, G'_t)$$

- The number of iterations T is a hyperparameter of the kernel that you have to fix in advance or cross-validate

W-L Graph kernel: Remarks

- Concatenating feature vectors is the same as summing kernels so effectively we have mapped graphs into a sequence of graphs $G_0 = G, G_1, \ldots, G_T$ and computed

$$K_{\text{WL}}(G, G') = \sum_{t=0}^{T} k(G_t, G'_t)$$

- The number of iterations T is a hyperparameter of the kernel that you have to fix in advance or cross-validate

- The "base kernel" k may be more complicated than just counting the number of common colors, e.g. could use shortest paths

$$K_{\text{WL-sp}}(G, G') = \sum_{t=0}^{T} k_{\text{sp}}(G_t, G'_t)$$

W-L Graph kernel: Results (CPU time)

Data Set	MUTAG	NCI1	NCI109	ENZYMES	D & D
Maximum # nodes	28	111	111	126	5748
Average # nodes	17.93	29.87	29.68	32.63	284.32
# labels	7	37	38	3	82
Number of graphs	188	4110	4127	600	1178
WL subtree	6”	7’20”	7’21”	20”	11’0”
WL edge	3”	1’5”	58”	11”	3 days
WL shortest path	2”	2’20”	2’23”	1’3”	484 days
Ramon & Gärtner	40’6”	81 days	81 days	38 days	103 days
p-random walk	4’42”	5 days	5 days	10’	4 days
Random walk	12”	9 days	9 days	12’19”	48 days
Graphlet count	3”	1’27”	1’27”	25”	30’21”
Shortest path	2”	4’38”	4’39”	5”	23h 17’2”

Table 2: CPU runtime for kernel computation on graph classification benchmark data sets

W-L Graph kernel: Results (accuracy)

Method/Data Set	MUTAG	NCI1	NCI109	ENZYMES	D & D
WL subtree	82.05 ($\pm$0.36)	82.19 ($\pm$ 0.18)	82.46 ($\pm$0.24)	52.22 ($\pm$1.26)	79.78 ($\pm$0.36)
WL edge	81.06 ($\pm$1.95)	84.37 ($\pm$0.30)	84.49 ($\pm$0.20)	53.17 ($\pm$2.04)	77.95 ($\pm$0.70)
WL shortest path	83.78 ($\pm$1.46)	84.55 ($\pm$0.36)	83.53 ($\pm$0.30)	59.05 ($\pm$1.05)	79.43 ($\pm$0.55)
Ramon & Gärtner	85.72 ($\pm$0.49)	61.86 ($\pm$0.27)	61.67 ($\pm$0.21)	13.35 ($\pm$0.87)	57.27 ($\pm$0.07)
p-random walk	79.19 ($\pm$1.09)	58.66 ($\pm$0.28)	58.36 ($\pm$0.94)	27.67 ($\pm$0.95)	66.64 ($\pm$0.83)
Random walk	80.72 ($\pm$0.38)	64.34 ($\pm$0.27)	63.51 ($\pm$ 0.18)	21.68 ($\pm$0.94)	71.70 ($\pm$0.47)
Graphlet count	75.61 ($\pm$0.49)	66.00 ($\pm$0.07)	66.59 ($\pm$0.08)	32.70 ($\pm$1.20)	78.59 ($\pm$0.12)
Shortest path	87.28 ($\pm$0.55)	73.47 ($\pm$0.11)	73.07 ($\pm$0.11)	41.68 ($\pm$1.79)	78.45 ($\pm$0.26)

Table 1: Prediction accuracy ($\pm$ standard deviation) on graph classification benchmark data sets

Neighborhood Subgraph Pairwise Distance Kernel

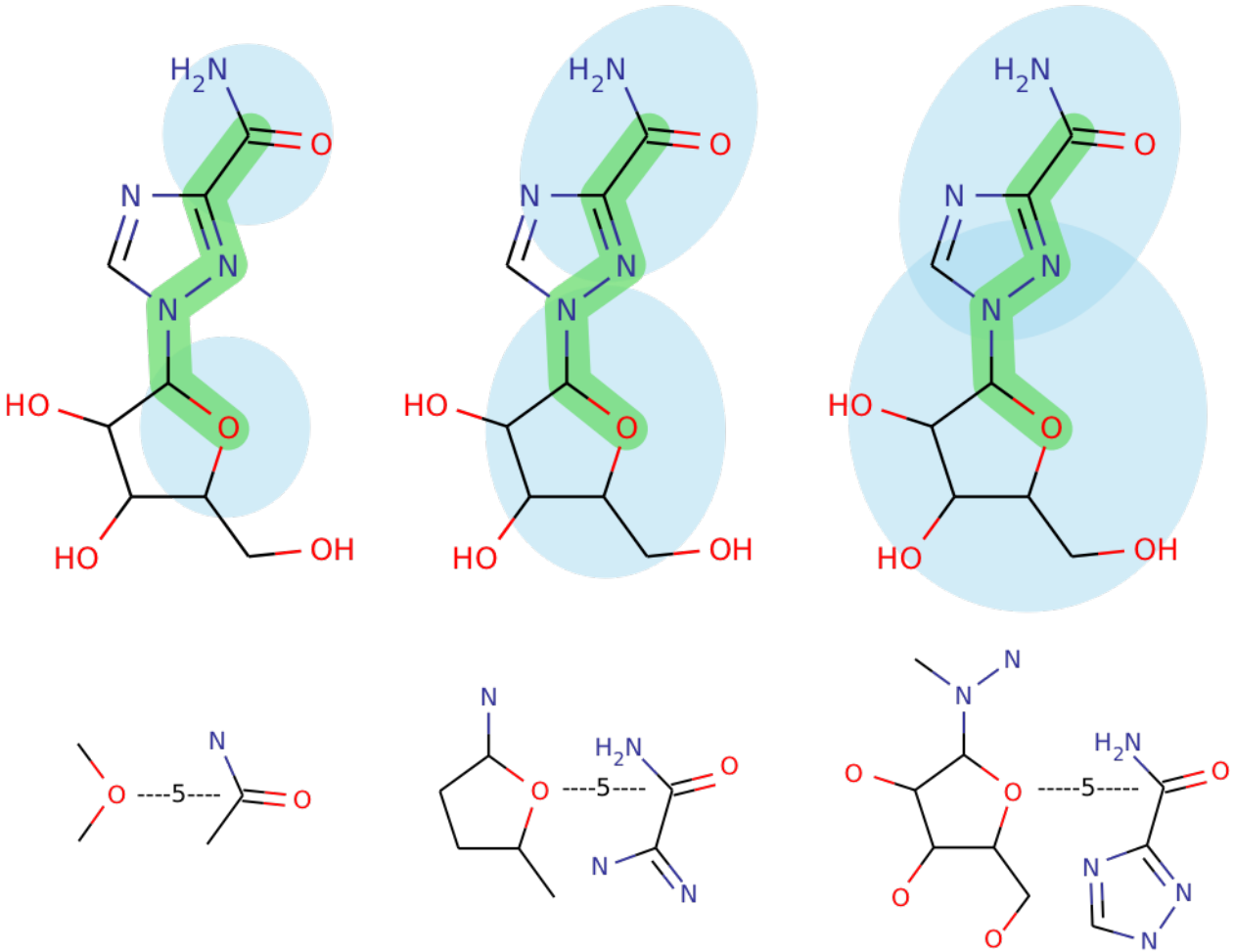
- Convolution kernel based on the relation (Costa et al. 2010)

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

where

- $\delta_{u,v}$ is the shortest-path distance between u and v
- the neighborhood $\mathcal{N}_r^v(G)$ is the subgraph of G induced by all $u \in V$ s.t. $\delta_{u,v} \leq r$

Neighborhood Subgraph Pairwise Distance Kernel



Pairs of neighborhood graphs for radius $r = 1,2,3$ and distance $d = 5$

Neighborhood Subgraph Pairwise Distance Kernel

- $\kappa_{r,d}$ counts the # of common neighborhood subgraphs:

$$\kappa_{r,d}(G, G') = \sum_{\substack{(A,B) \in R_{r,d}^{-1}(G) \\ (A',B') \in R_{r,d}^{-1}(G')}} \mathbb{1}\{A \approx A'\}\,\mathbb{1}\{B \approx B'\}$$

Neighborhood Subgraph Pairwise Distance Kernel

- $\kappa_{r,d}$ counts the # of common neighborhood subgraphs:

$$\kappa_{r,d}(G, G') = \sum_{\substack{(A,B) \in R_{r,d}^{-1}(G) \\ (A',B') \in R_{r,d}^{-1}(G')}} \mathbb{1}\{A \approx A'\}\,\mathbb{1}\{B \approx B'\}$$

- Hashing used to map subgraphs into IDs — somewhat related to (Weinberger et al. 2009)

Neighborhood Subgraph Pairwise Distance Kernel

- $\kappa_{r,d}$ counts the # of common neighborhood subgraphs:

$$\kappa_{r,d}(G, G') = \sum_{\substack{(A,B) \in R_{r,d}^{-1}(G) \\ (A',B') \in R_{r,d}^{-1}(G')}} \mathbb{1}\{A \approx A'\}\,\mathbb{1}\{B \approx B'\}$$

- Hashing used to map subgraphs into IDs — somewhat related to (Weinberger et al. 2009)

- Overall kernel:

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

NSPKD Results (CPU time)

Table 2. Net CPU time of graph kernels in seconds

		NCI-60	HIV	PTC	Bursi
	# of mol.	3910	42687	417	4337
	Aug. time	$3.5 \cdot 10^2$	$3.4 \cdot 10^3$	$3.4 \cdot 10^1$	$1.2 \cdot 10^2$
Graph Fragment Kernel (Wale et al. 2008)	GFK(G)	$3.5 \cdot 10^1$	$1.4 \cdot 10^4$	$3.1 \cdot 10^0$	$7.3 \cdot 10^1$
Weighted Decomposition Kernel (Menchetti et al. 2005)	WDK(G)	$1.8 \cdot 10^3$	$1.6 \cdot 10^5$	$8.0 \cdot 10^0$	$1.1 \cdot 10^3$
	WDK(G_a)	$2.3 \cdot 10^3$	$2.3 \cdot 10^5$	$1.4 \cdot 10^1$	$1.5 \cdot 10^3$
Pairwise Maximum Common Subgraphs Kernel (Schietgat et al. 2011)	PMCSK(G)	$2.8 \cdot 10^5$	$3.3 \cdot 10^{4*}$	$6.2 \cdot 10^2$	$3.5 \cdot 10^5$
	PMCSK(G'_a)	$2.8 \cdot 10^5$	$3.3 \cdot 10^{4*}$	$6.3 \cdot 10^2$	$3.5 \cdot 10^5$
Shortest-path Kernel (Shervashidze & Borgwardt, 2009)	PDK(G)	$4.2 \cdot 10^1$	$3.9 \cdot 10^3$	$1.0 \cdot 10^0$	$3.6 \cdot 10^1$
	PDK(G_a)	$7.7 \cdot 10^1$	$4.2 \cdot 10^3$	$2.0 \cdot 10^0$	$5.7 \cdot 10^1$
	NSK(G)	$6.2 \cdot 10^1$	$3.1 \cdot 10^3$	$2.8 \cdot 10^0$	$5.1 \cdot 10^1$
	NSK(G_a)	$3.5 \cdot 10^2$	$6.0 \cdot 10^3$	$1.4 \cdot 10^1$	$2.0 \cdot 10^2$
	NSPDK(G)	$1.2 \cdot 10^2$	$1.0 \cdot 10^4$	$3.4 \cdot 10^0$	$1.1 \cdot 10^2$
	NSPDK(G_a)	$4.6 \cdot 10^2$	$1.9 \cdot 10^4$	$1.6 \cdot 10^1$	$2.9 \cdot 10^2$

* MCSs derived only from the 1503 CA-CM molecules.

NSPKD Results (accuracy)

Table 1. Generalization performance of kernels on unaugmented and augmented molecular graphs

	NCI-60 (avg.)	HIV CA vs. CM	HIV CACM vs. CI	HIV CA vs. CI	PTC (avg.)	Bursi
AUROC (%)						
GFK(G)	77.8 ± 2.3	82.0 ± 4.7	82.8 ± 1.9	93.9 ± 2.6	62.6 ± 10	89.6 ± 0.3
WDK(G)	71.1 ± 2.4	83.1 ± 4.3	82.9 ± 1.8	94.0 ± 3.4	62.1 ± 7.7	88.0 ± 0.4
WDK(G_a)	**80.0 ± 2.3**	84.2 ± 4.3	83.9 ± 1.7	95.0 ± 2.7	65.1 ± 8.7	90.8 ± 0.2
PMCSK(G)	79.6 ± 2.2	**82.6 ± 6.2**	81.8 ± 2.2	93.0 ± 3.7	64.5 ± 8.8	90.5 ± 1.3
PMCSK(G'_a)	**80.3 ± 2.2**	**82.8 ± 6.2**	**83.2 ± 2.1**	93.4 ± 3.4	65.6 ± 8.8	**91.5 ± 1.1**
PDK(G)	73.4 ± 2.6	81.6 ± 4.6	77.7 ± 1.9	92.6 ± 3.2	61.2 ± 9.7	82.7 ± 0.3
PDK(G_a)	77.8 ± 2.4	82.1 ± 4.2	83.4 ± 2.1	94.5 ± 2.4	64.6 ± 9.9	89.3 ± 0.3
NSK(G)	79.1 ± 2.2	**84.2 ± 4.9**	84.3 ± 2.0	95.3 ± 1.5	67.4 ± 9.4	91.6 ± 0.2
NSK(G_a)	79.4 ± 2.2	**84.4 ± 4.5**	84.1 ± 2.2	94.9 ± 2.1	67.1 ± 9.3	**91.8 ± 0.2**
NSPDK(G)	79.5 ± 2.2	**83.9 ± 5.6**	83.8 ± 2.1	**95.6 ± 1.3**	**69.3 ± 9.5**	**91.7 ± 0.3**
NSPDK(G_a)	**80.1 ± 2.2**	**84.1 ± 4.8**	**84.9 ± 2.1**	**95.1 ± 2.0**	**68.9 ± 9.8**	**92.0 ± 0.2**

Part 4

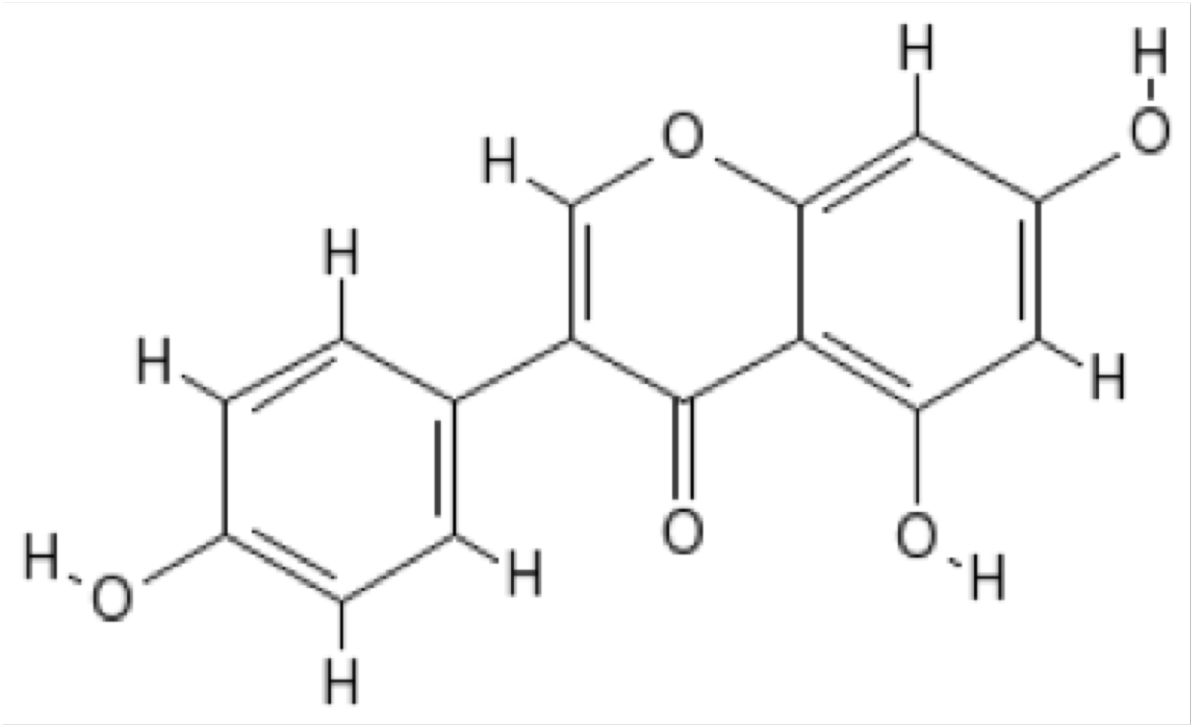
Kernel methods for relational learning

kFOIL (Landwehr et al. 2006, 2010)
kLog (Frasconi et al. 2014)

- Kernel methods in the ILP setting

- Kernel methods in the ILP setting
- Very simple idea:
  - Define a kernel on relational data based on a relational theory
  - Perform structure learning to induce the relational theory
  - This effectively learns the kernel

Relational data example



Examples

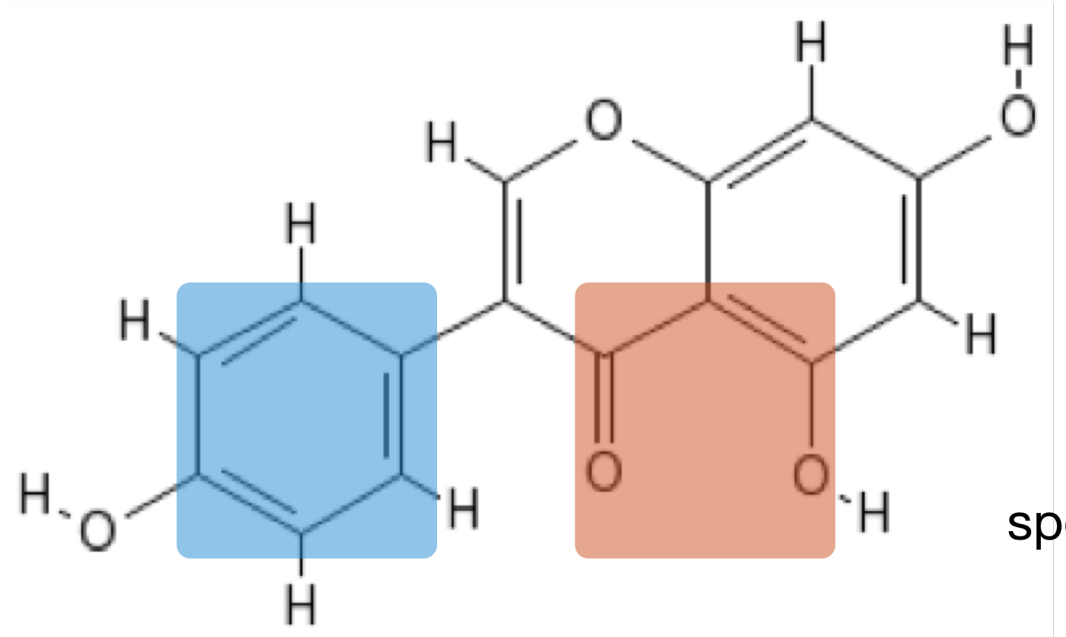
```
pos(m1).
neg(m2).
pos(m3).
pos(m4).
…
```

Background knowlegde

```
molecule(m1).
atom(m1,a11,c).
atom(m1,a12,c).
bond(m1,a11,a12,1).
charge(m1,a11,0.82).
…
```

```
aromatic_ring(M) :-
  bond(M,A,B,7),bond(M,B,C,7),
  bond(M,C,D,7),bond(M,D,E,7),
  bond(M,E,F,7),bond(M,F,A,7).
```

Relational data + theory = features



Clauses match on specific structural features

Theory: set of clauses

```
c1(M,X) :-
  bond(M,A,B,7),bond(M,B,C,7),
  bond(M,C,D,7),bond(M,D,E,7),
  bond(M,E,F,7),bond(M,F,A,7).
```

```
c2(M,X) :-
  atom(M,A,o),bond(M,A,B,2),
  atom(M,B,c),bond(M,B,C,1),
  atom(M,C,c),bond(M,C,D,7),
  atom(M,D,c),bond(M,D,E,1),
  atom(M,E,o).
```

Relational data + theory = features and kernel

- Let $H = \{c_1, \ldots, c_p\}$ be the theory (set of clauses)

Relational data + theory = features and kernel

- Let $H = \{c_1, \ldots, c_p\}$ be the theory (set of clauses)
- Let x be one example and let $\phi(x)$ be the feature vector defined as

$$\phi_j(x) = \begin{cases} 1 & \text{if } c_j \text{ fires on } x \\ 0 & \text{otherwise} \end{cases}$$

Relational data + theory = features and kernel

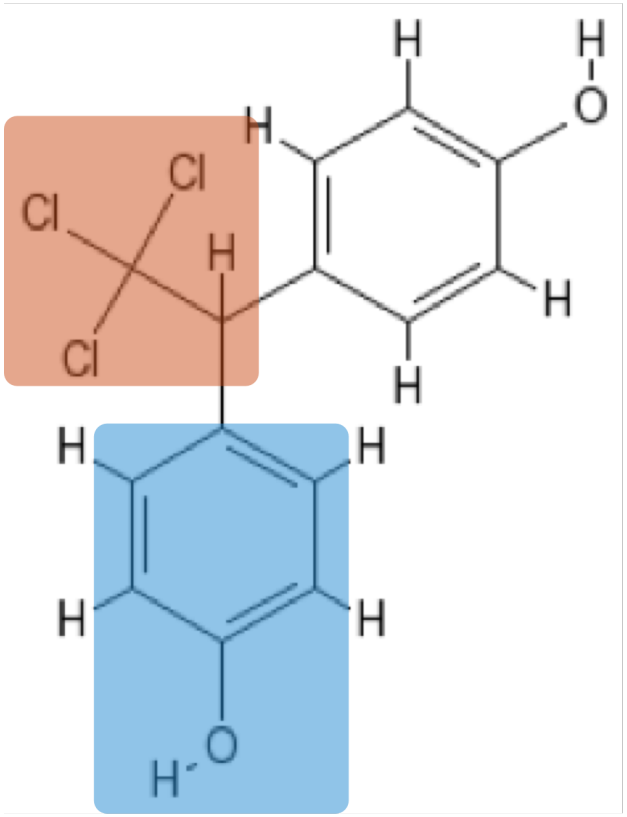
- Let $H = \{c_1, \ldots, c_p\}$ be the theory (set of clauses)
- Let x be one example and let $\phi(x)$ be the feature vector defined as

$$\phi_j(x) = \begin{cases} 1 & \text{if } c_j \text{ fires on } x \\ 0 & \text{otherwise} \end{cases}$$

- The kernel is of course

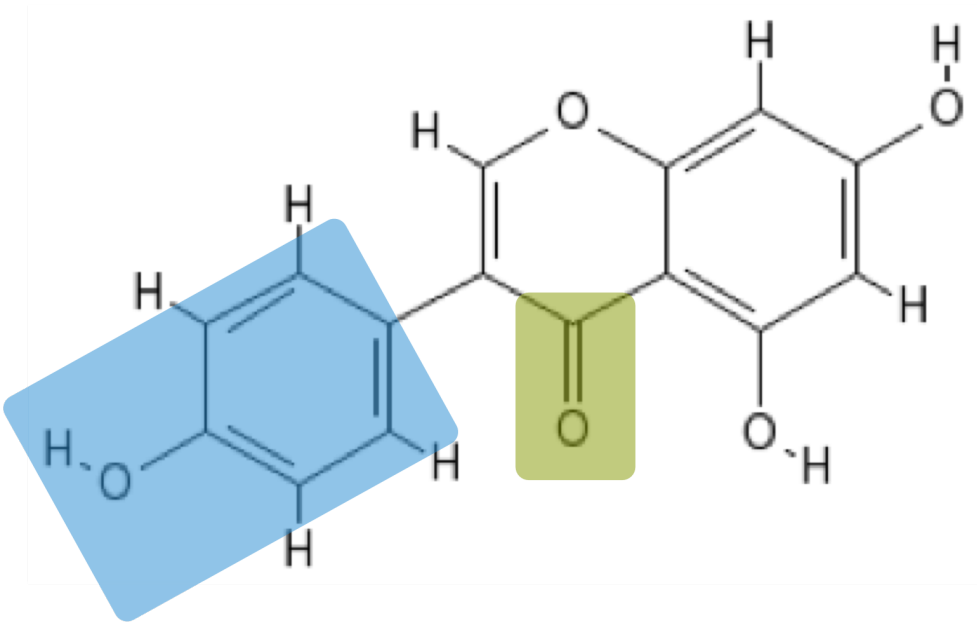
$$k(x, z) = \langle \phi(x), \phi(z) \rangle$$

Example



$\phi(x) = [1, 1, 0]$

Clauses: c1, c2, c3



$\phi(z) = [0, 1, 1]$

$$k(x, z) = \langle [1, 1, 0], [0, 1, 1] \rangle = 1$$

- FOIL style greedy general-to-specific search for clauses

- FOIL style greedy general-to-specific search for clauses
- Refinement operator: Given a clause c, refine it into c' by finding a minimal specialization in the language of clauses

Searching

- FOIL style greedy general-to-specific search for clauses
- Refinement operator: Given a clause c, refine it into c' by finding a minimal specialization in the language of clauses
- In traditional ILP the goal is to find a theory that covers all positive examples and no negative example

- FOIL style greedy general-to-specific search for clauses
- Refinement operator: Given a clause c, refine it into c' by finding a minimal specialization in the language of clauses
- In traditional ILP the goal is to find a theory that covers all positive examples and no negative example
- In kFOIL the goal is to maximize the accuracy score of a kernel machine based on the kernel defined earlier

- A framework and domain specific language for kernel-based relational learning

- A framework and domain specific language for kernel-based relational learning
- Embedded in Prolog

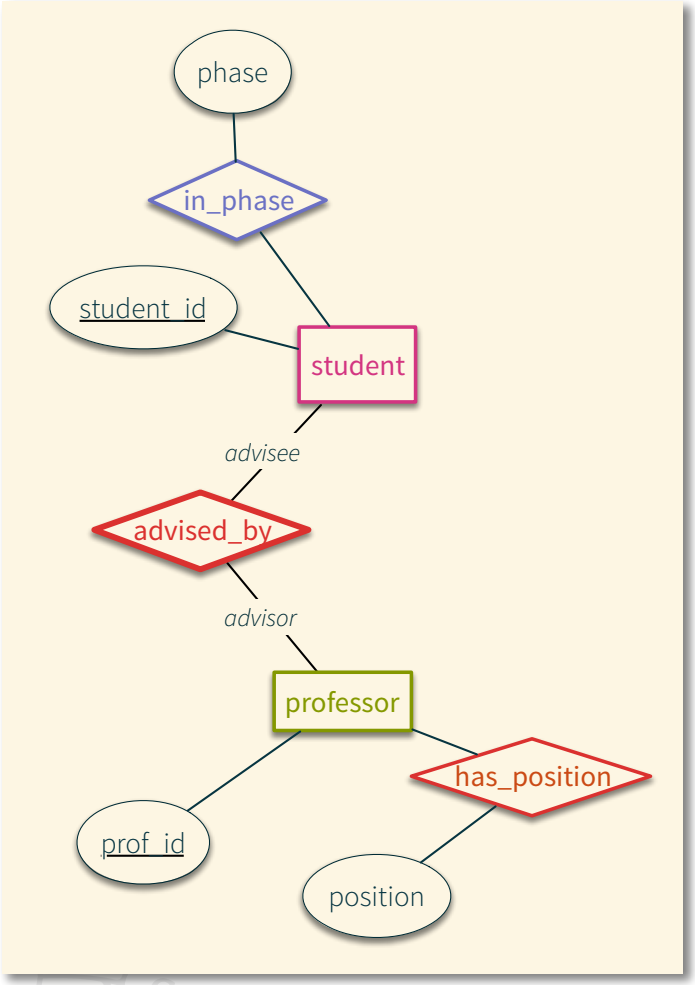
- A framework and domain specific language for kernel-based relational learning
- Embedded in Prolog
- Three simple concepts:
  1. Entity/relationship (E/R) data modeling combined with ideas from *deductive databases*
  2. Graphicalization: Examples (i.e. relational database instances) mapped to (simple) undirected graphs
  3. Graph kernels: construction of feature vectors from graphs

- A framework and domain specific language for kernel-based relational learning
- Embedded in Prolog
- Three simple concepts:
  1. Entity/relationship (E/R) data modeling combined with ideas from *deductive databases*
  2. Graphicalization: Examples (i.e. relational database instances) mapped to (simple) undirected graphs
  3. Graph kernels: construction of feature vectors from graphs
- Available: `http://klog.dinfo.unifi.it/`

- Design and maintain complex features in a declarative fashion

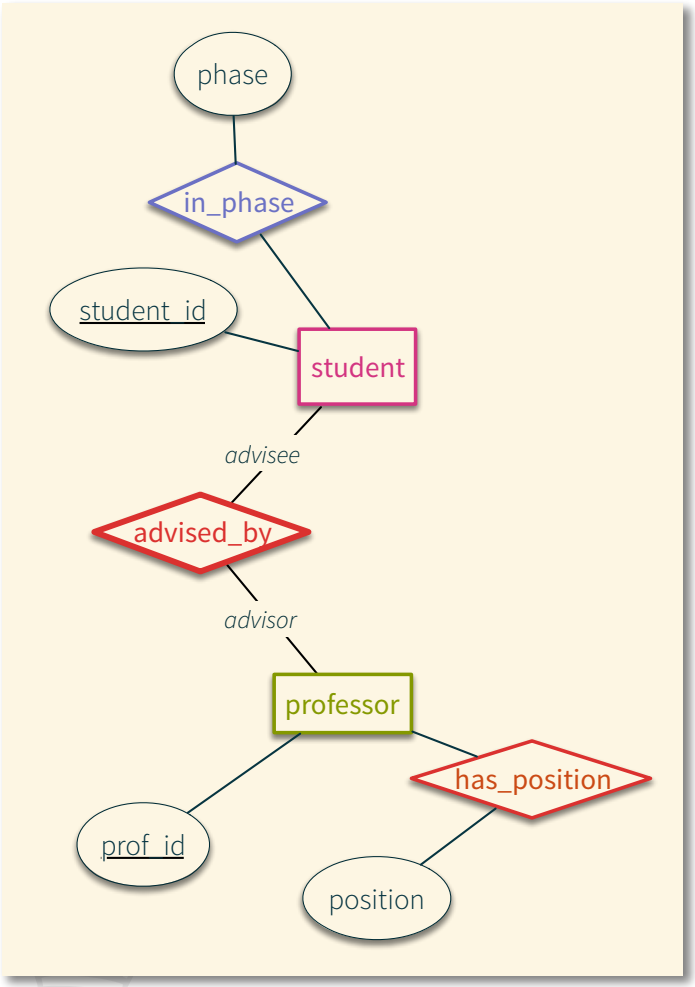
- Design and maintain complex features in a declarative fashion
- Ability to specify several kinds of learning problems, including:
  - classification/regression of structured data
  - entity classification
  - (hyper)link prediction

Modeling the UW-CSE dataset in kLog



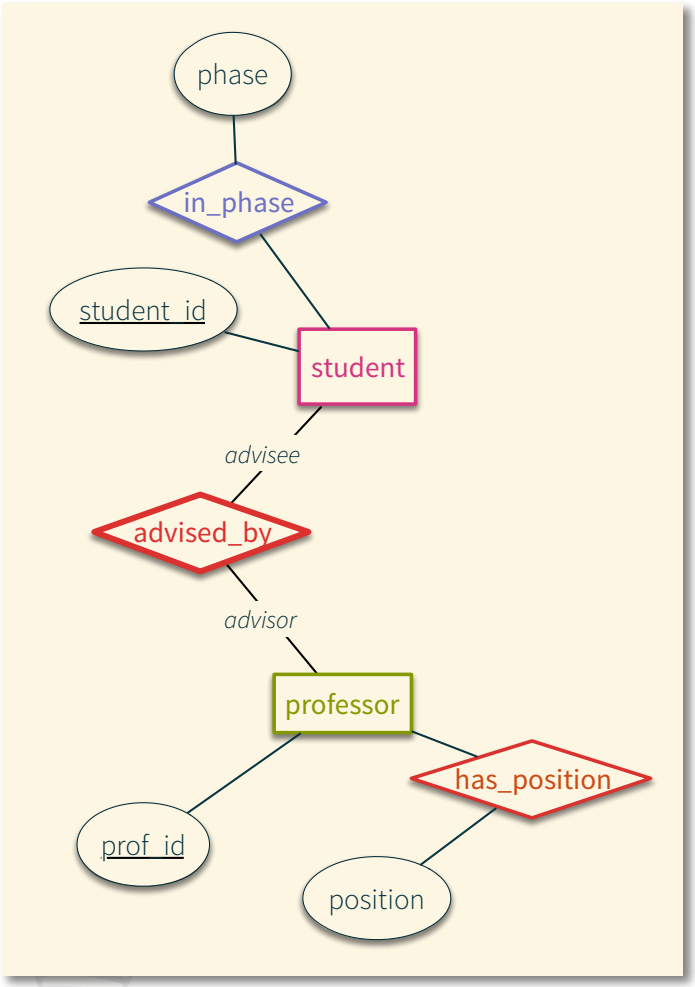
- Introduced in (Richardson & Domingos, 2005) to illustrate Markov logic
- Entity/Relationship (E/R) diagram:
  - Boxes are entities
  - Diamonds are relationships
  - Ovals are attributes
  - Underlined attributes are entity identifiers, the other ones are properties

Modeling the UW-CSE dataset in kLog




```
signature student(
    student_id::self
)::extensional.
signature in_phase(
    student_id::student,
    phase::property)::extensional.

signature professor(
   prof_id::self
)::extensional.
signature has_position(
    prof_id::professor,
    position::property
)::extensional.
```

Modeling the UW-CSE dataset in kLog

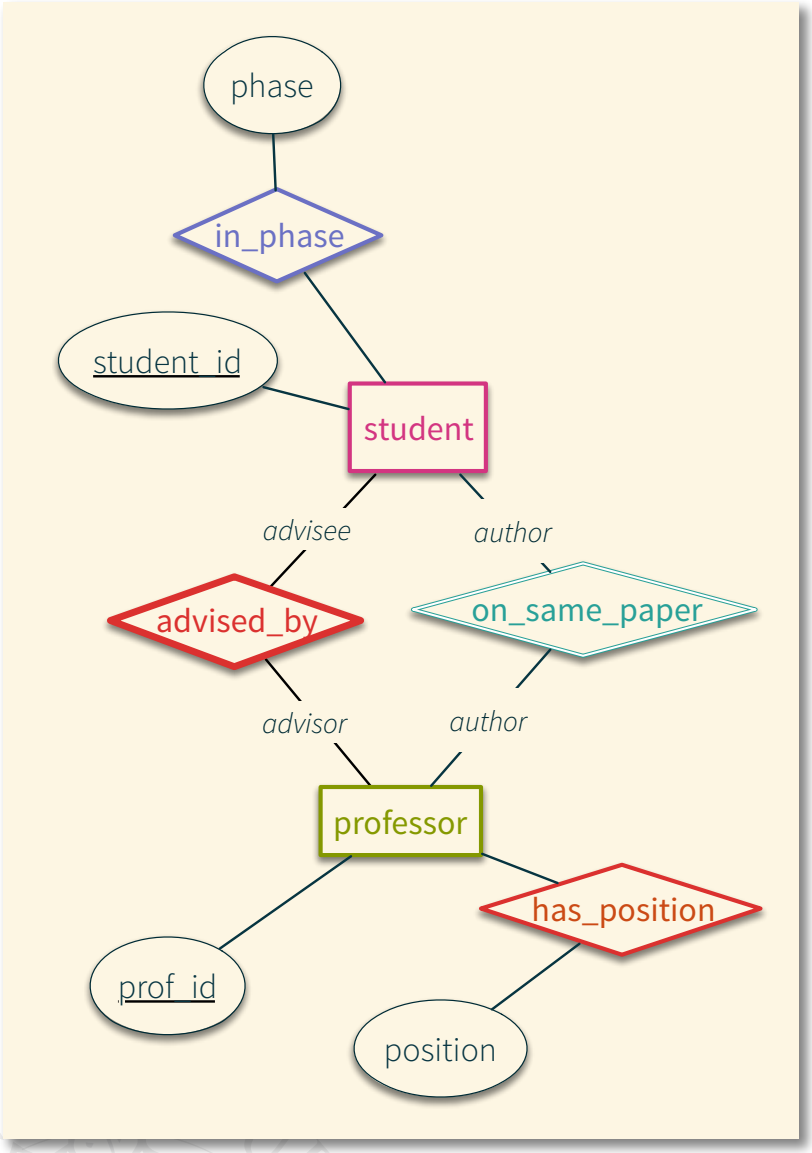



```
signature student(
    student_id::self
)::extensional.
signature in_phase(
    student_id::student,
    phase::property)::extensional.

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.
```

Learning from interpretations: predictors and responses

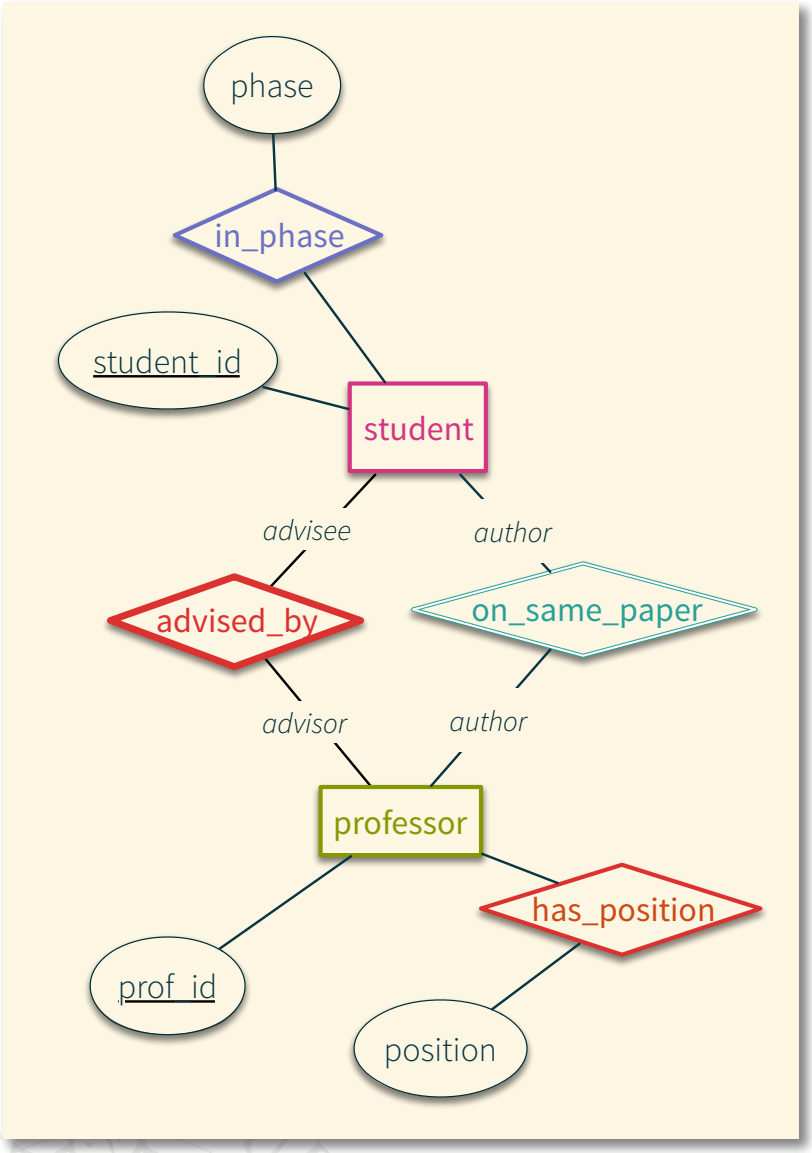
```
student(person311).
student(person14).
...
professor(person7).
professor(person185).
...
has_position(person292,faculty_affiliate).
has_position(person79,faculty).
...
in_phase(person139,post_quals).
in_phase(person333,pre_quals).
...
advised_by(person265,person168).
advised_by(person352,person415).
...
publication(title25,person284).
...
taught_by(course12,person211,autumn_0001).
...
ta(course44,person193,winter_0304).
...
publication(title25,person284).
```

Adding background knowledge: Intensional signatures



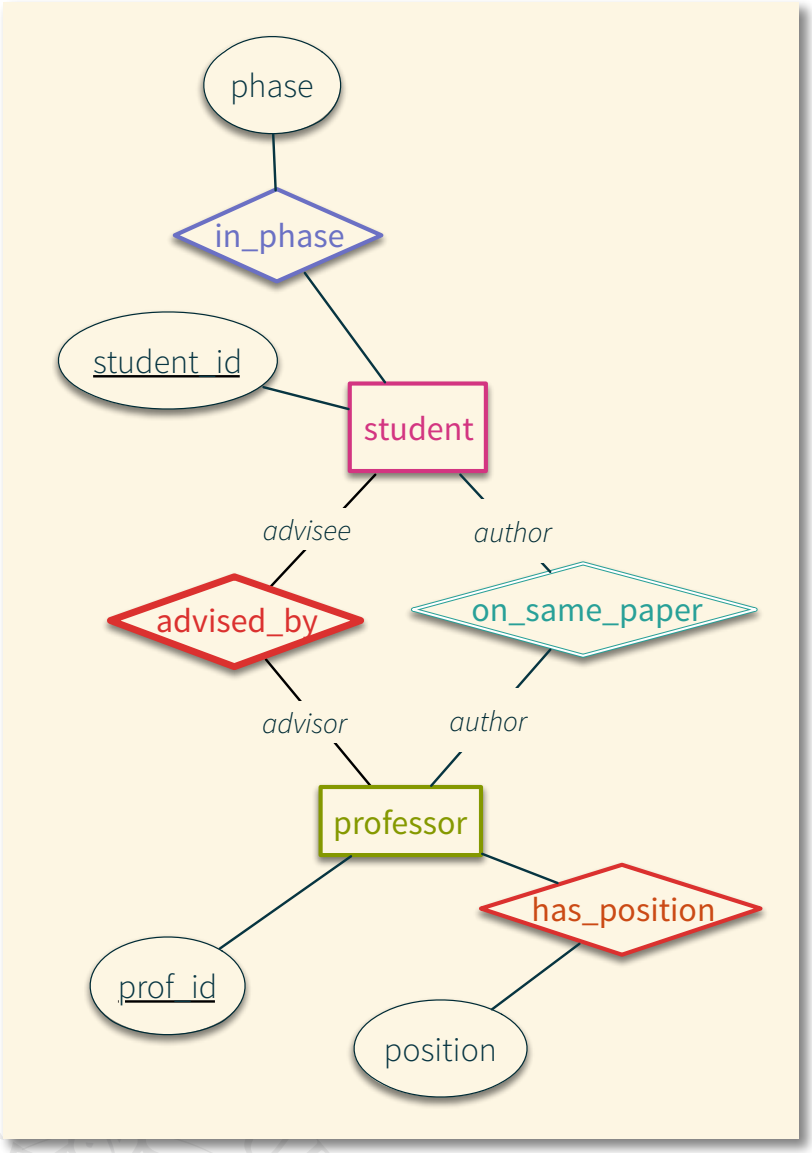

```
signature on_same_paper(
    student_id::student,
    prof_id::professor
)::intensional.
```

Adding background knowledge: 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).
```

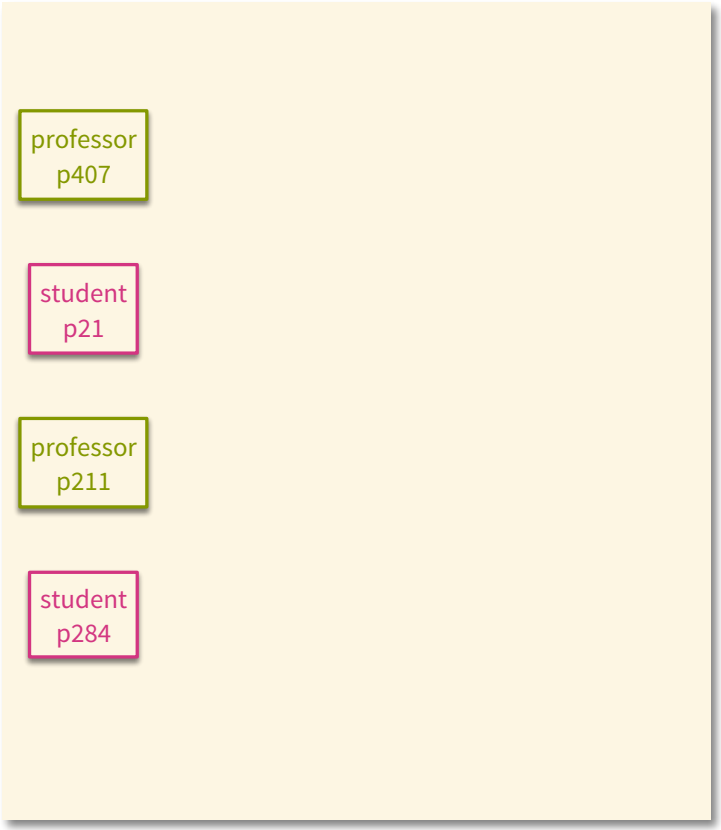
Adding background knowledge: Intensional signatures




```
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).
```

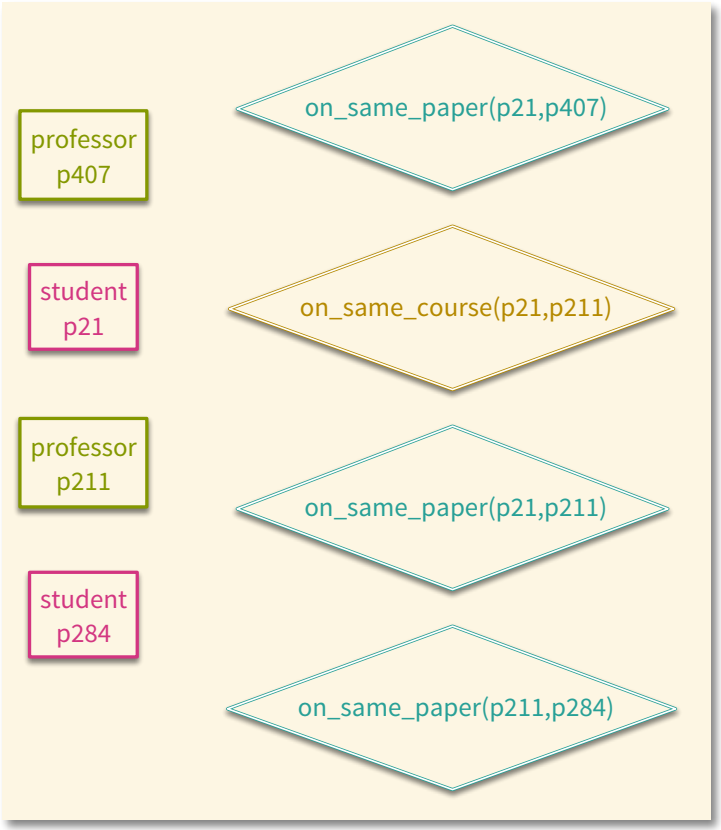
```
signature on_same_course(
   student_id::student,
   prof_id::professor
)::intensional.

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

Graphicalization: from interpretations to bipartite graphs



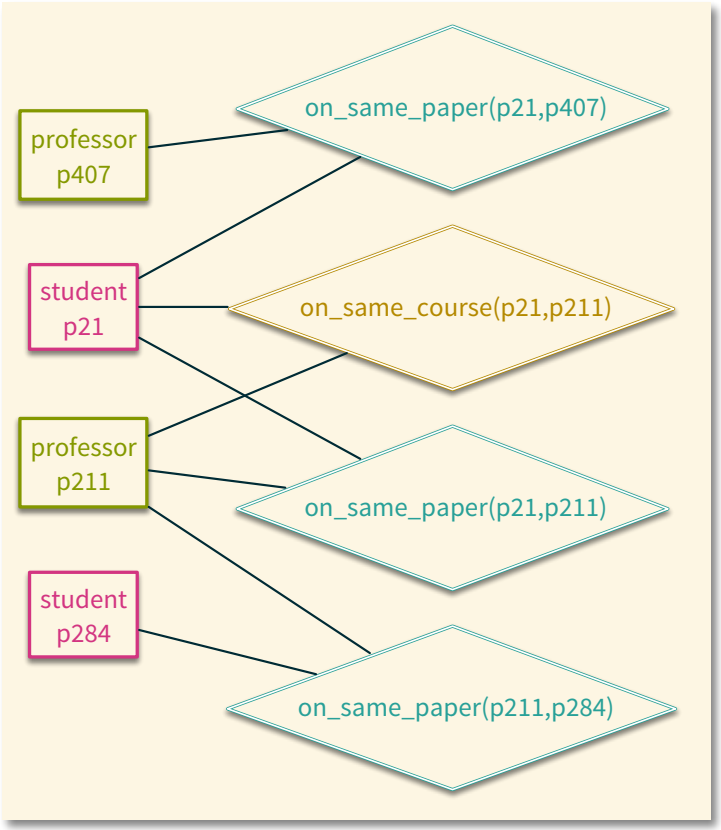
- One square vertex for every entity

Graphicalization: from interpretations to bipartite graphs



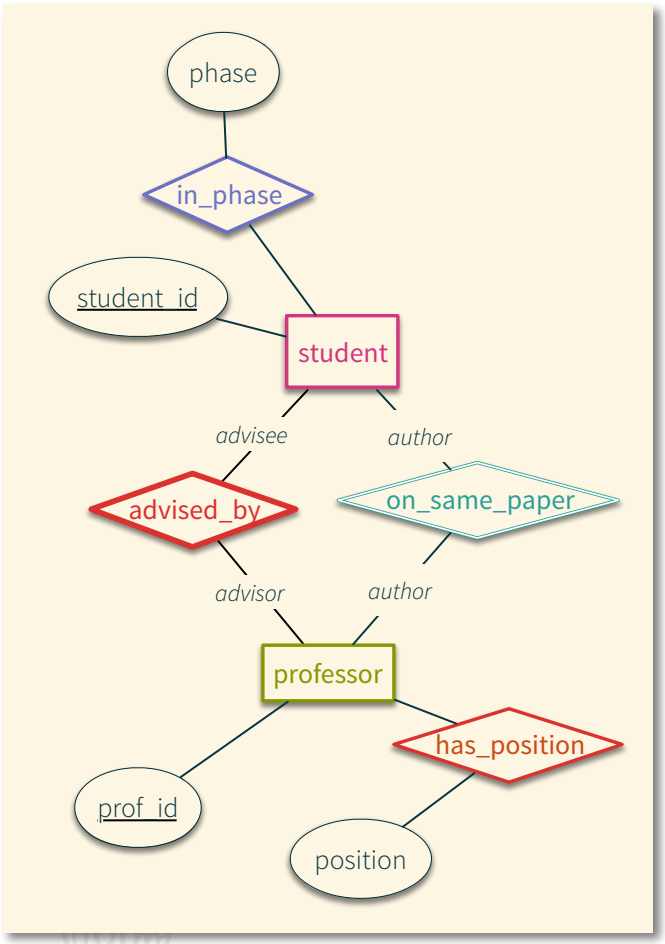
- One square vertex for every entity
- One diamond vertex for every ground relationship

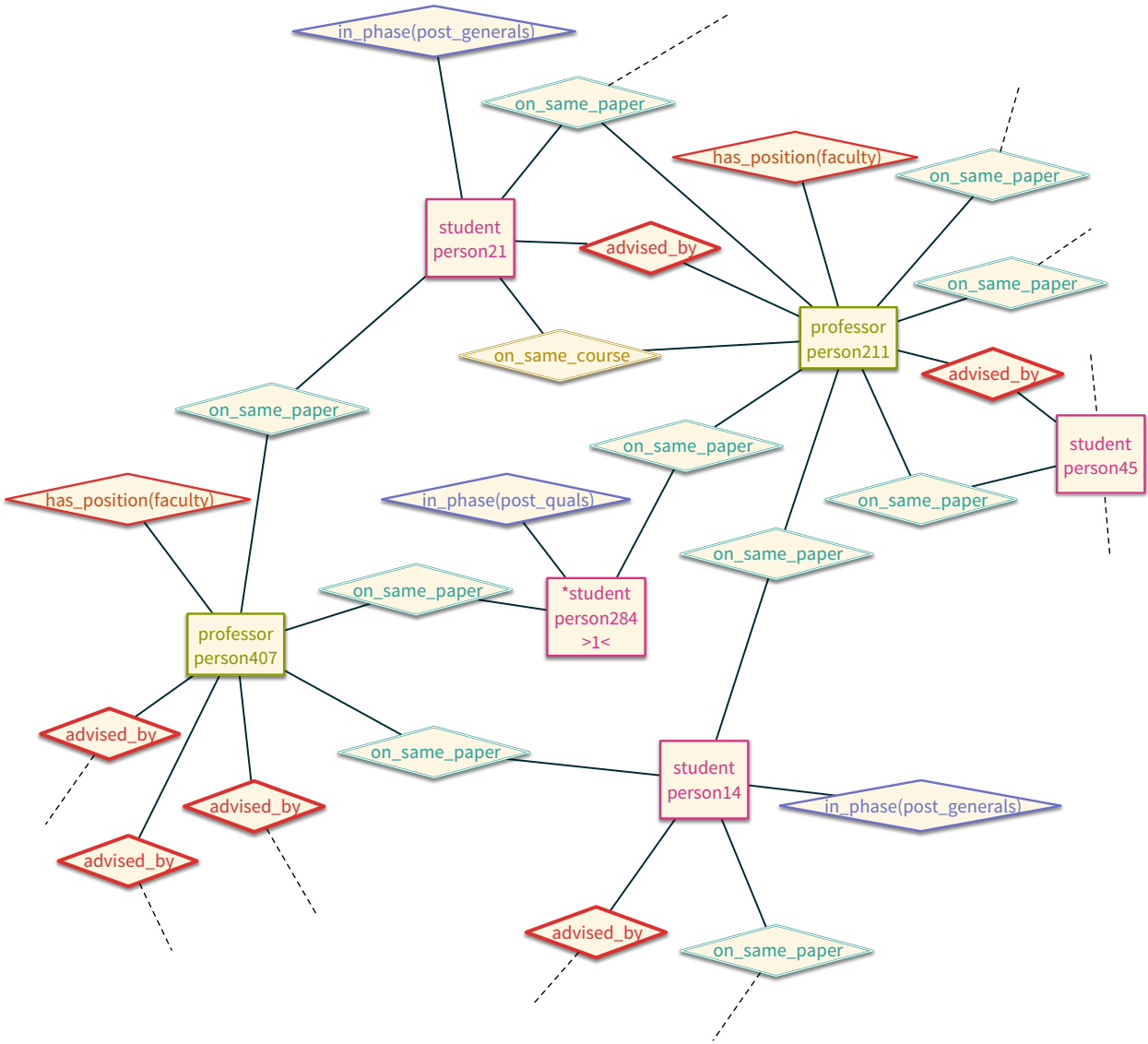
Graphicalization: from interpretations to bipartite graphs



- One square vertex for every entity
- One diamond vertex for every ground relationship
- Add an undirected edge between a square and a diamond if the entity appears in the grounding

Graphicalization in UW-CSE





Using graph kernels to construct features

- In principle, any graph kernel may be adapted and plugged in

Using graph kernels to construct features

- In principle, any graph kernel may be adapted and plugged in
- In practice, kLog uses a generalization of NSPDK (Costa et al. 2010) where:
  - Subgraphs are rooted at certain designated vertices called kernel-points (KP)
  - Soft matches are allowed

Soft matches

- Substructures will never exactly match if there are "hubs" or high-degree vertices

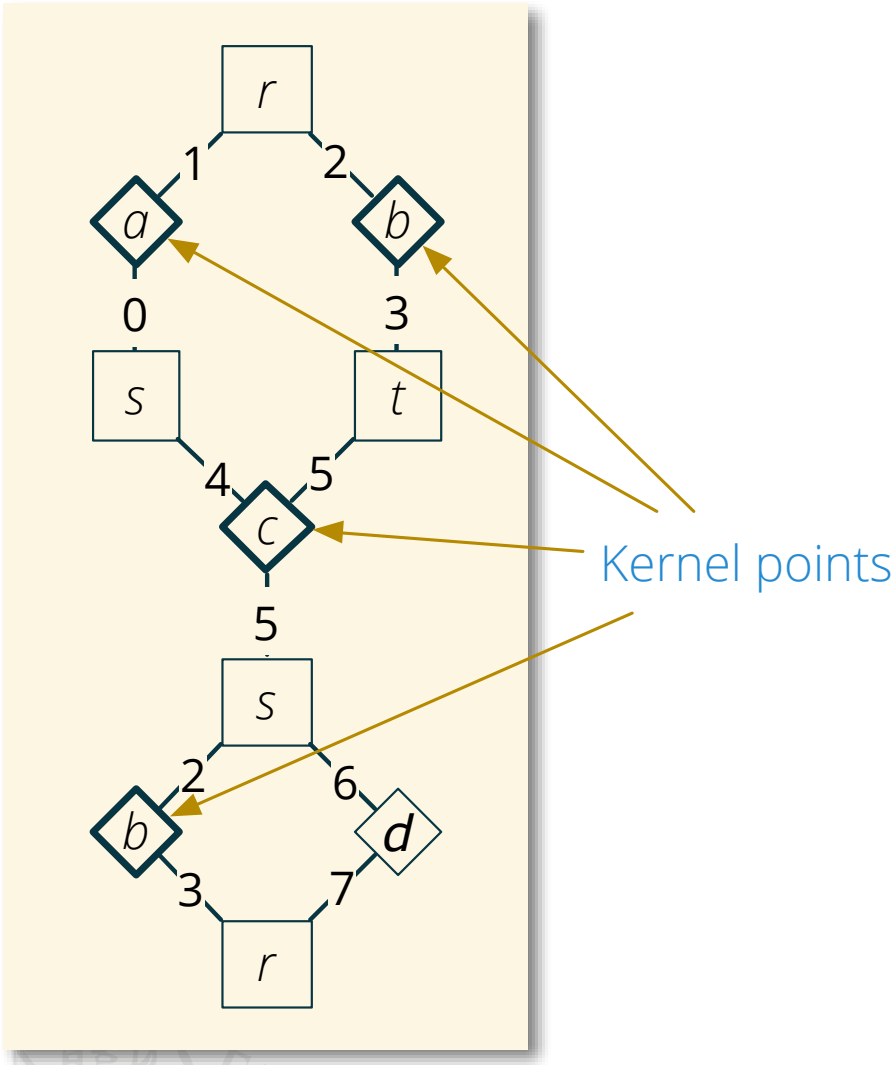
Soft matches

- Substructures will never exactly match if there are "hubs" or high-degree vertices
- Example: the relation `has_word` between words and webpages

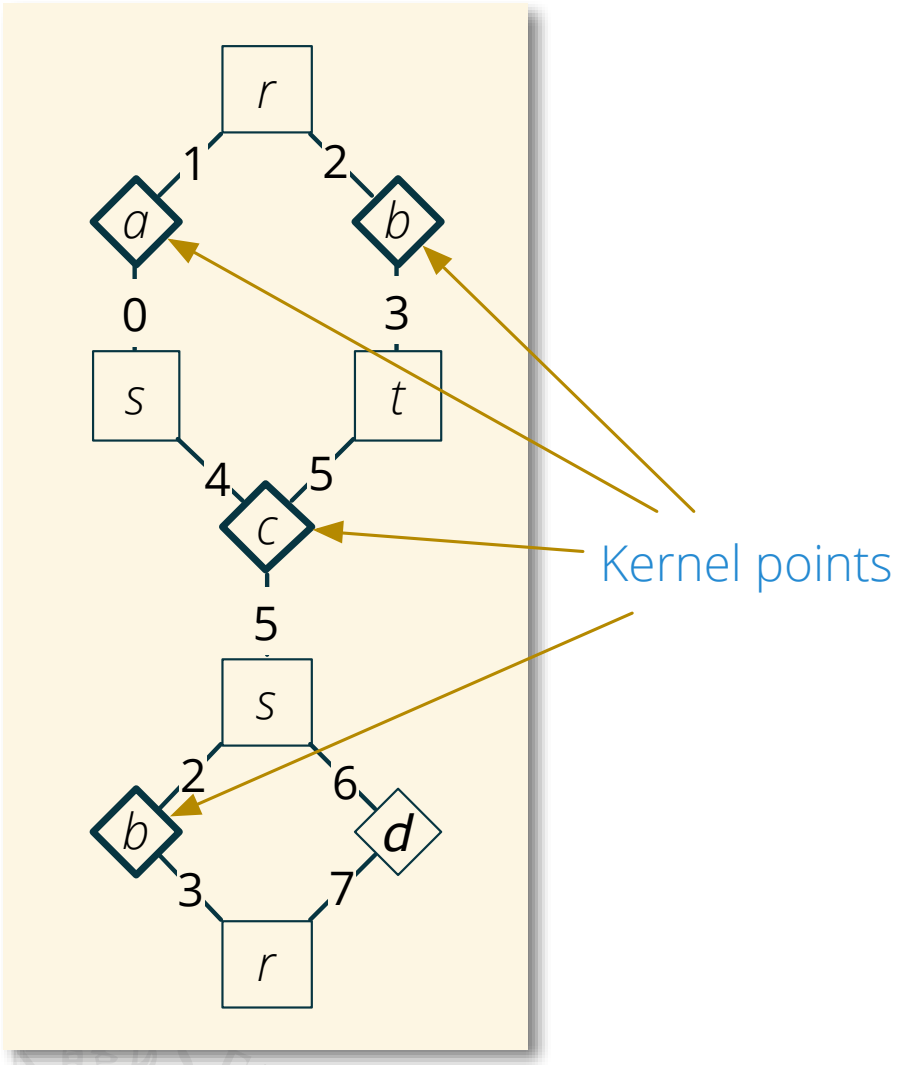
- Substructures will never exactly match if there are "hubs" or high-degree vertices
- Example: the relation `has_word` between words and webpages
- Soft match kernel:

$$\kappa_{r,d}(G, G') = \sum_{\substack{(A,B) \in R_{r,d}^{-1}(G) \\ (A',B') \in R_{r,d}^{-1}(G')}} \sum_{\substack{v \in V(A) \cup V(B) \\ v' \in V(A') \cup V(B')}} \mathbb{1}\{\ell(v) = \ell(v')\}$$

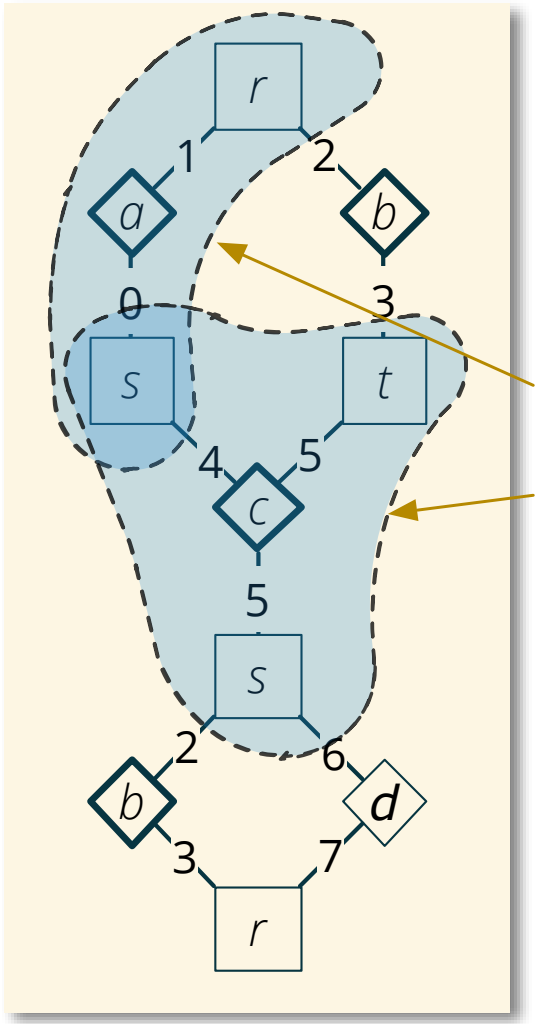
Kernel details



Kernel details

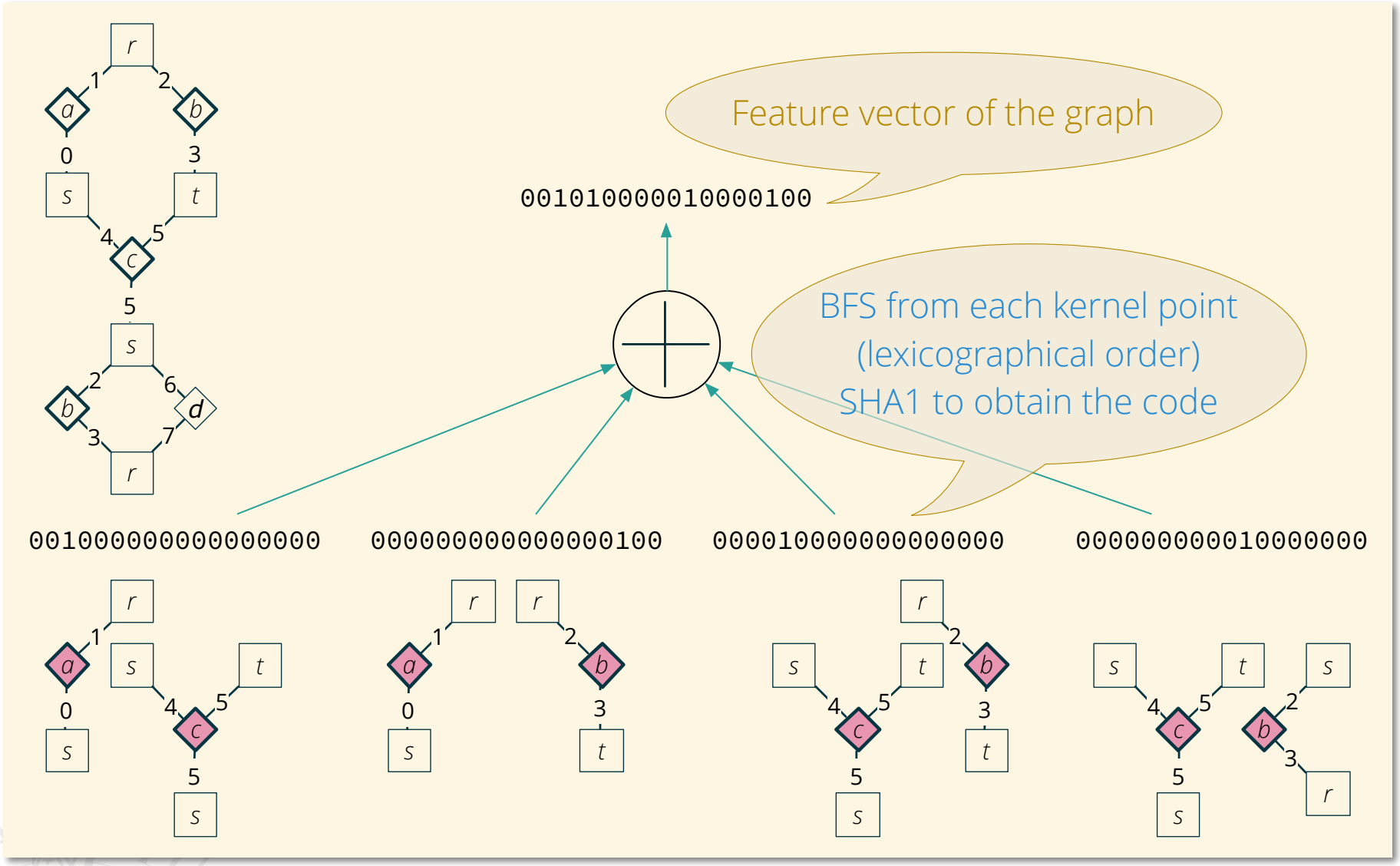


Kernel points



A neighborhood pair
radius=1
distance=2

Kernel details



- Let x and y denote the sets of input ground atoms (predictors) and output ground atoms (responses).

- Let x and y denote the sets of input ground atoms (predictors) and output ground atoms (responses).
- Graphicalization and feature generation yields a joint feature vector $\phi(x, y)$

- Let x and y denote the sets of input ground atoms (predictors) and output ground atoms (responses).
- Graphicalization and feature generation yields a joint feature vector $\phi(x, y)$
- Fit a linear *potential* function:

$$F(x, y) = w^\top \phi(x, y)$$

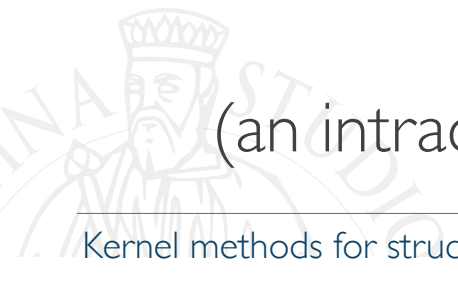
- Let x and y denote the sets of input ground atoms (predictors) and output ground atoms (responses).
- Graphicalization and feature generation yields a joint feature vector $\phi(x, y)$
- Fit a linear *potential* function:

$$F(x, y) = w^\top \phi(x, y)$$

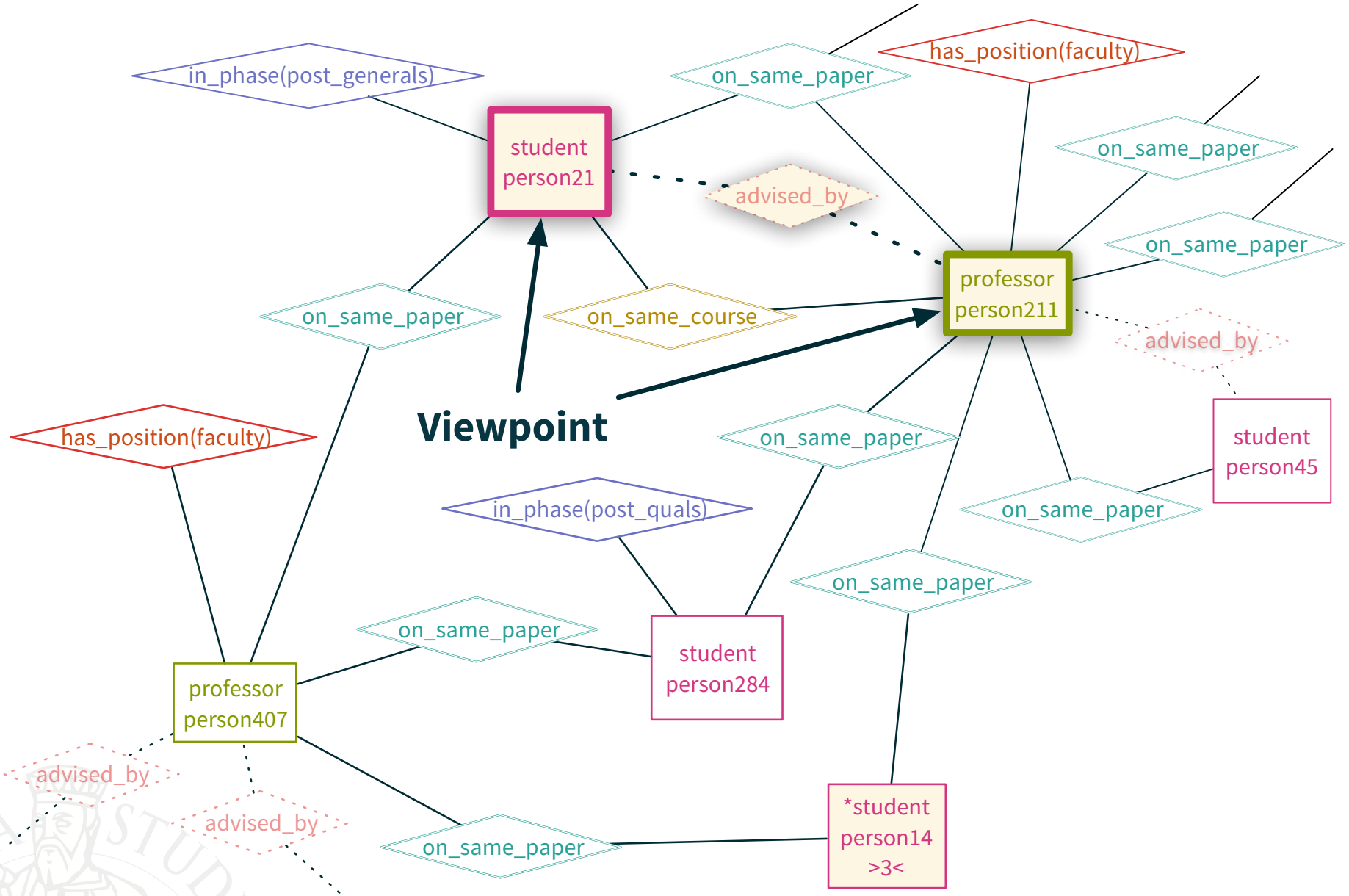
- Prediction: solve the "inference" problem

$$f(x) = \arg\max_y F(x, y)$$

(an intractable step, in general)



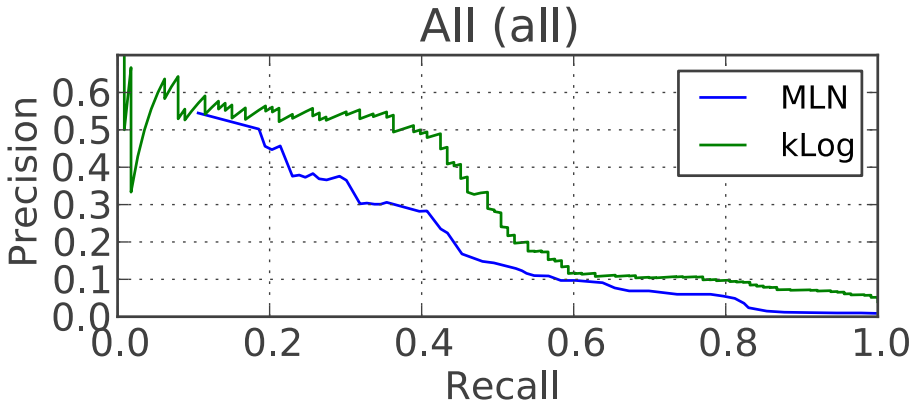
Viewpoints example

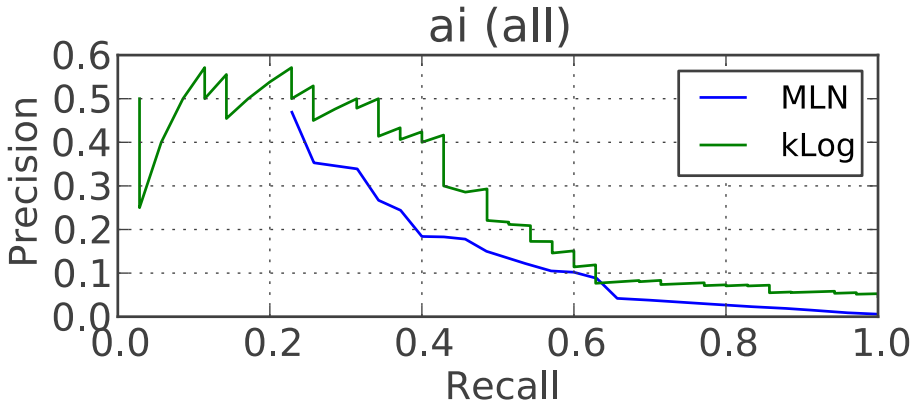


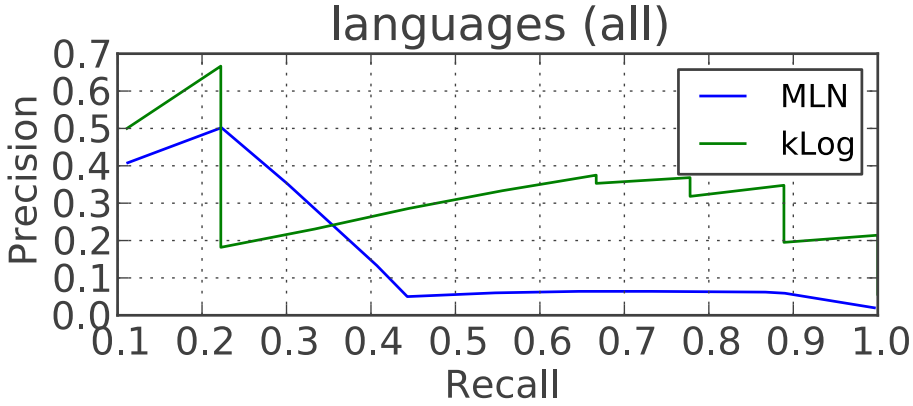
A whole kLog script

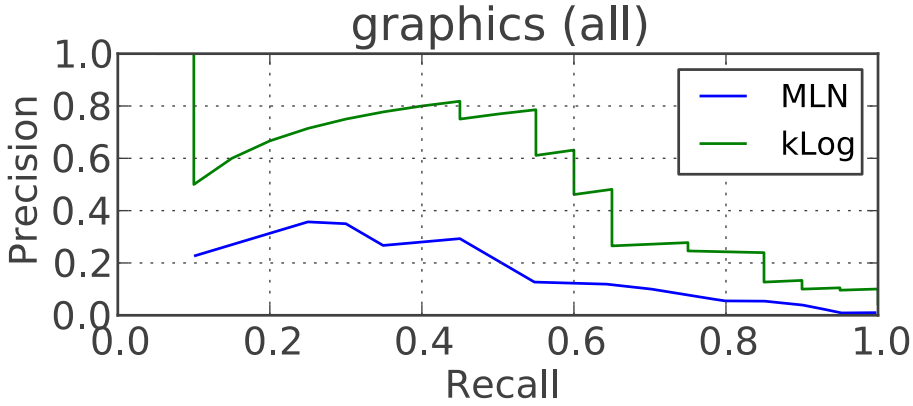
```
:- use_module('klog').
begin_domain.
  signature student(student_id::self)::extensional.
  signature professor(professor_id::self)::extensional.
  signature on_same_course(s::student,p::professor)::intensional.
  on_same_course(S,P) :-
      professor(P), student(S), ta(C,S,Term), taught_by(C,P,Term).
  signature on_same_paper(s::student,p::professor)::intensional.
  on_same_paper(S,P) :-
      student(S), professor(P), publication(Pub, S), publication(Pub,P).
  signature advised_by(s_id::student,p_id::professor)::extensional.
  kernel_points([student,professor,on_same_course,on_same_paper]).
end_domain.

experiment :-
   new_feature_generator(my_fg,nspdk),
   set_klog_flag(my_fg,radius,2),
   set_klog_flag(my_fg,distance,2),
   attach(uwcse_ext),
   new_model(my_model,svm_sgd),
   set_klog_flag(my_model,lambda,0.0001),
   set_klog_flag(my_model,epochs,5), %% ... etc
   kfold(advised_by,5,my_model,my_fg).
```

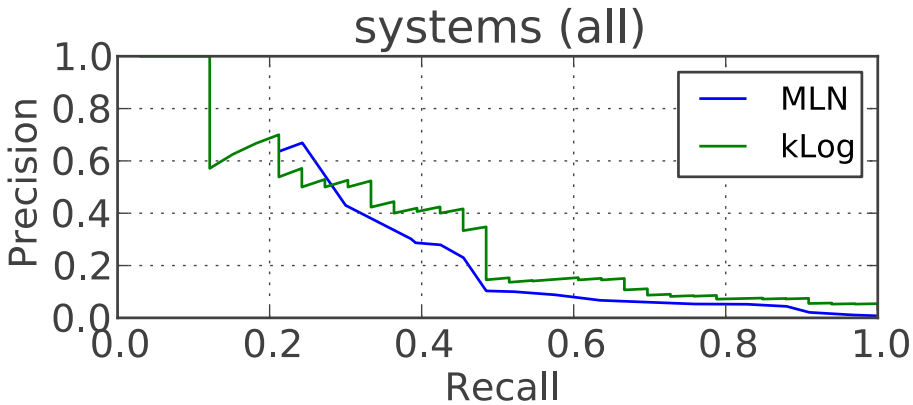
Example: UW-CSE (All information)

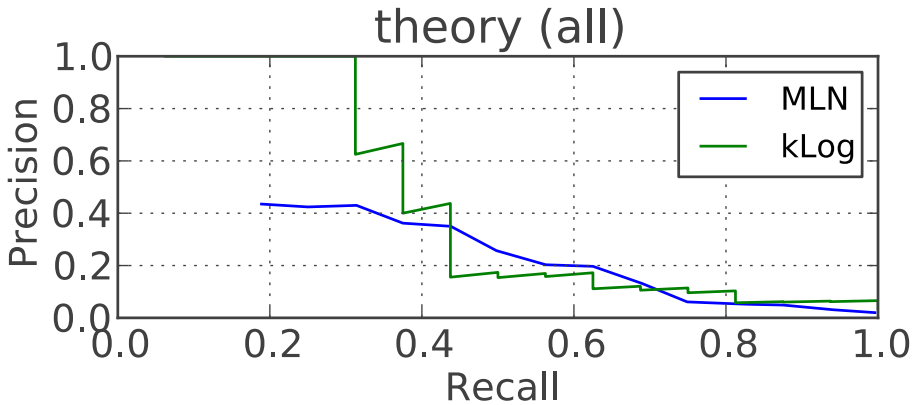












Alternative setting: partial information

- We only know about *persons* without knowing whether they are professors or students

Alternative setting: partial information

- We only know about *persons* without knowing whether they are professors or students
- Stacking in kLog

Alternative setting: partial information

- We only know about *persons* without knowing whether they are professors or students
- Stacking in kLog
  - First, learn to discriminate between professors and students

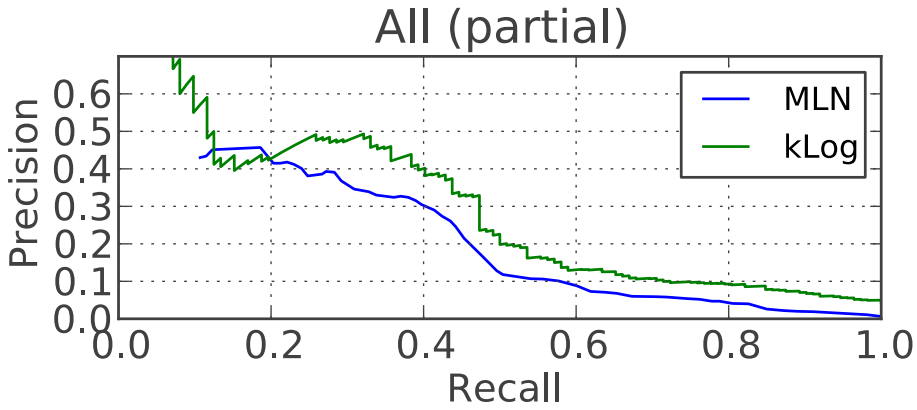
Alternative setting: partial information

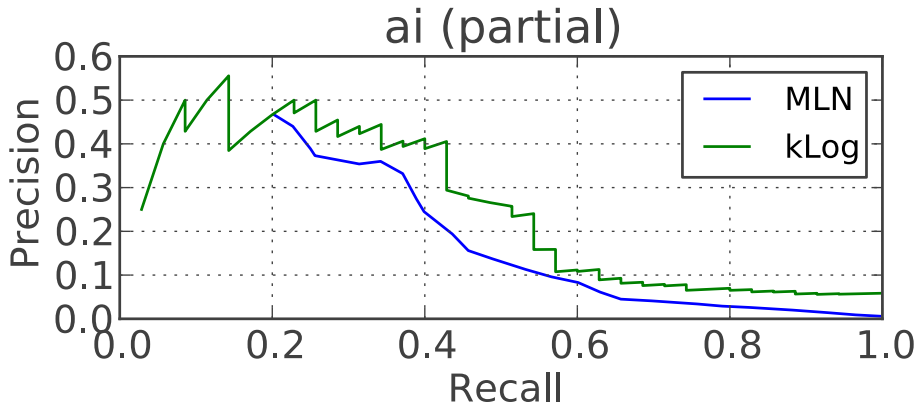
- We only know about *persons* without knowing whether they are professors or students
- Stacking in kLog
  - First, learn to discriminate between professors and students
  - Assert *induced* groundings (predicted in cross-validation mode)

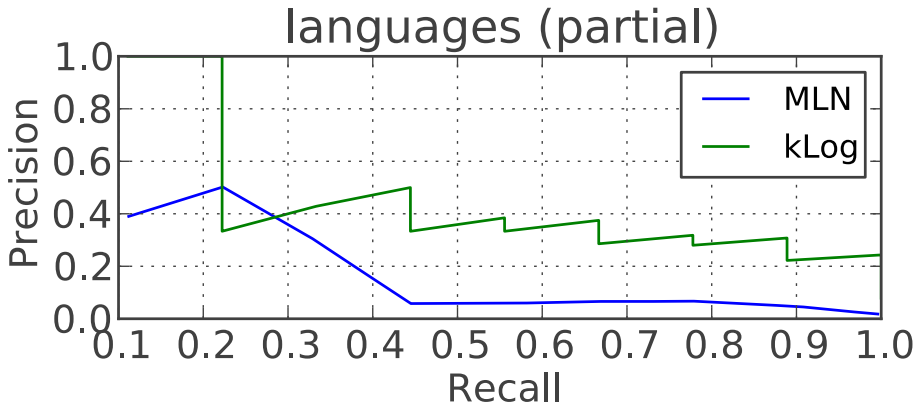
Alternative setting: partial information

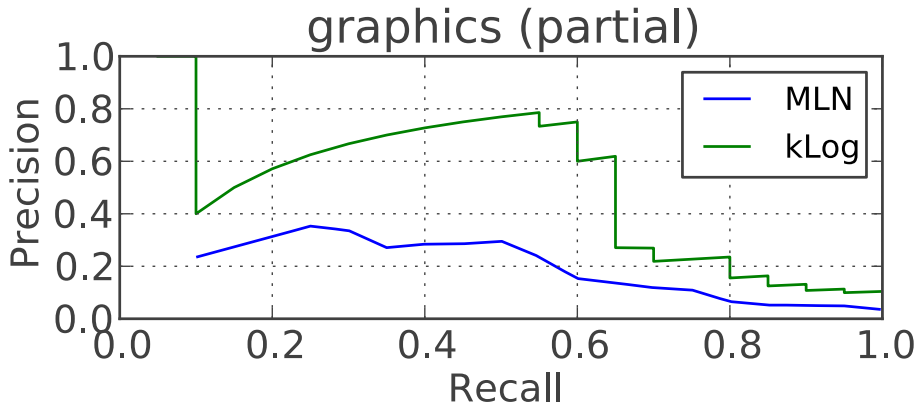
- We only know about *persons* without knowing whether they are professors or students
- Stacking in kLog
  - First, learn to discriminate between professors and students
  - Assert *induced* groundings (predicted in cross-validation mode)
  - Learn the binary relation taking saved groundings as additional data

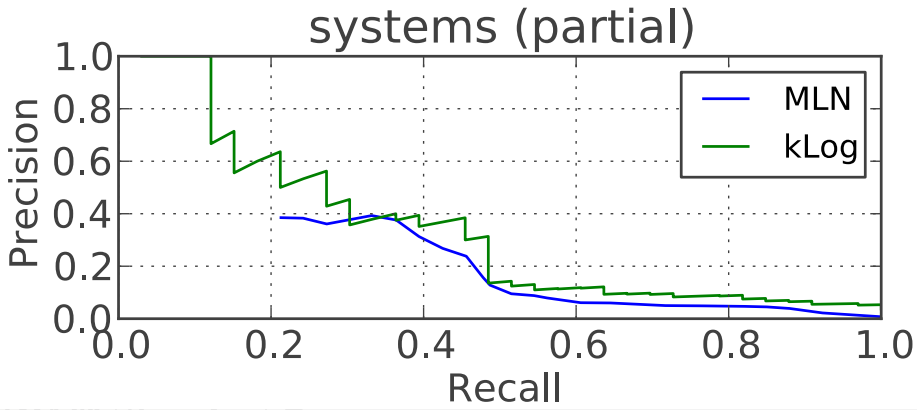
Example: UW-CSE (Partial information)

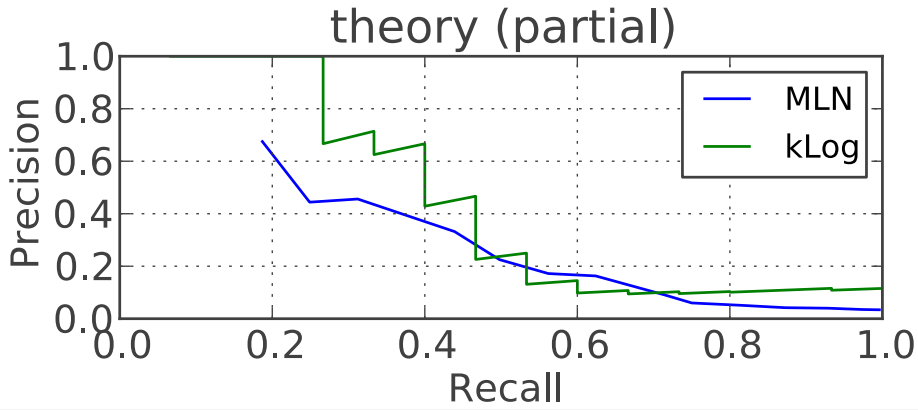












WebKB: results

	kLog		Markov Logic		Tilde	
	Acc	F_1	Acc	F_1	Acc	F_1
research	94%	0.68	95%	0.66	93%	0.54
faculty	91%	0.74	92%	0.71	91%	0.71
course	99%	0.98	98%	0.95	99%	0.98
student	90%	0.91	89%	0.90	88%	0.89
Average	88%	0.88	88%	0.81	86%	0.78
Time	< 1m		450m		87m	

IMDb: results

Year	# Movies	# Facts	kLog	MLN	Tilde
1997	311	8031	0.86	0.79	0.80
1998	332	7822	0.93	0.85	0.88
1999	348	7842	0.89	0.85	0.85
2000	381	8531	0.96	0.86	0.93
2001	363	8443	0.95	0.86	0.91
2002	370	8691	0.93	0.87	0.89
2003	343	7626	0.95	0.88	0.87
2004	371	8850	0.95	0.87	0.87
2005	388	9093	0.92	0.84	0.83
All			0.93 ± 0.03	0.85 ± 0.03	0.87 ± 0.04
Time			1,394s	220s	12,812s

- Natural language processing:
  - Hedge cue detection (Verbeke et al. 2011)
  - Evidence-based medicine (Verbeke et al. 2012)
- Vision:
  - Indoor scene classification (Antanas et al. 2013)

kLogNLP (Verbeke et al. 2014)

- Natural language module for kLog
- NLP-specific preprocessors, enabling the use of existing libraries, currently:
  - The Python Natural Language Toolkit (NLTK)
  - The Stanford CoreNLP
- `http://people.cs.kuleuven.be/~mathias.verbeke/klognlp/`

Hedge cue detection

- Hedge cues are linguistic devices that indicate whether information is being presented as uncertain or unreliable within a text
- Indicate caution or uncertainty towards content
- Task: discriminate between factual vs uncertain sentences, e.g.

Factual *"Among adolescents, the rate was found to be between 8 to 12 percent"*

Uncertain *"Some technologies are known to perform better than others in this regard"*

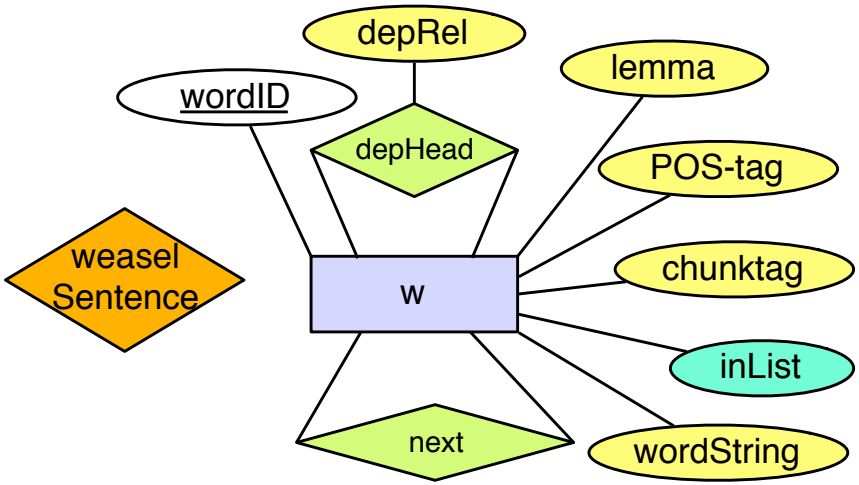
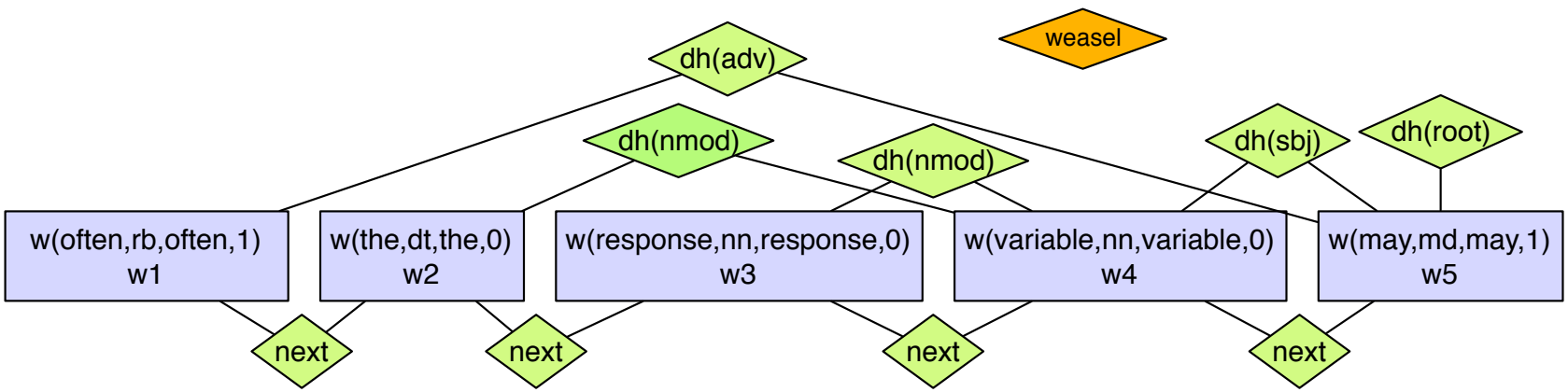


Fig. 2: E/R diagram modeling the hedge cue detection task



Results on CoNNL 2010 shared task (Verbeke et al. 2011)

Table 3: Evaluation performance in terms of precision, recall and F1 of the top 5 CoNLL 2010 systems and the kLog approach for the Wikipedia dataset

Official Rank	System	P	R	F
-	**kLog**	67.04	56.77	61.48
1	Georgescul	72.0	51.7	60.2
2	Ji[1]	62.7	55.3	58.7
3	Chen	68.0	49.7	57.4
4	Morante	80.6	44.5	57.3
5	Zhang	76.6	44.4	56.2

Part 5

Dealing with continuous/high dimensional attributes

Continuous and/or high-dimensional attributes

- Many kernels seen so far use hard-matching, which makes no sense in this setting

Continuous and/or high-dimensional attributes

- Many kernels seen so far use hard-matching, which makes no sense in this setting
- We will briefly review the following possible approaches:
  - Propagation kernels (Neumann et al. 2015, 2012)
  - GraphHopper (Feragen et al. 2013)
  - Graph invariant kernels (Orsini et al. 2015)

Propagation kernels

- Like in Weisfeiler-Lehman, define a sequence of graphs $G_0 = G, G_1, \ldots, G_T$ being T the number of propagation steps

- Like in Weisfeiler-Lehman, define a sequence of graphs $G_0 = G, G_1, \ldots, G_T$ being T the number of propagation steps

- As in W-L, the kernel between two graphs is

$$K(G, G') = \sum_{t=0}^{T} k(G_t, G'_t)$$

where

$$k(G_t, G'_t) = \sum_{v \in V_t} \sum_{v' \in V'_t} \kappa_{\text{node}}(v, v')$$

for some κ_{node} we will define later

- The propagation mechanism is based on the following diffusion process (Neumann et al. 2015):

$$P_{t+1} = TP_t$$

where

  - T is the row-normalized adjacency matrix
  - P_t contains a node distribution in each row

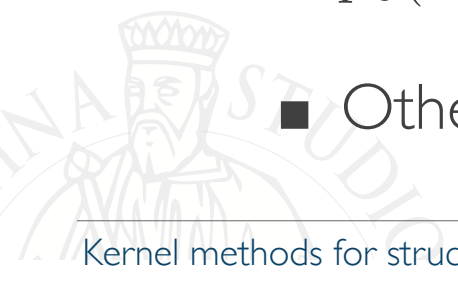
- The propagation mechanism is based on the following diffusion process (Neumann et al. 2015):

$$P_{t+1} = TP_t$$

where

  - T is the row-normalized adjacency matrix
  - P_t contains a node distribution in each row

- Initialization:
  - $p_0(v) = \delta_{\ell(v)}$ if v is labeled
  - Otherwise put a uniform distribution



Propagation kernels

- Distinguish between node labels $\ell(u)$ (categorical symbols) and node attributes $x(u)$ (may be real vectors)

- Distinguish between node labels $\ell(u)$ (categorical symbols) and node attributes $x(u)$ (may be real vectors)
- Given kernels κ_{label} and κ_{attr} for comparing labels and attributes, the node kernel is

$$\kappa_{\text{node}}(v, v') = \kappa_{\text{label}}(\ell(v), \ell(v'))\kappa_{\text{attr}}(x(v), x(v'))$$

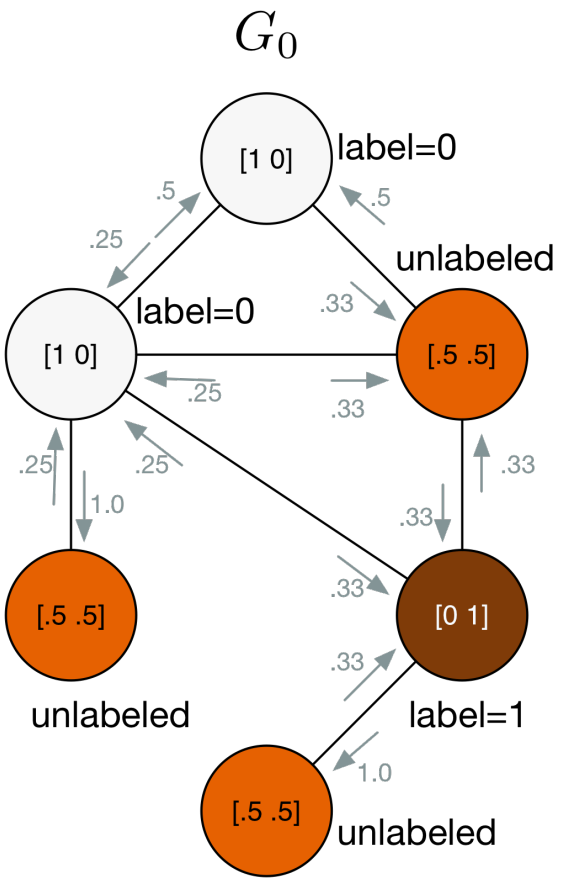
- Distinguish between node labels $\ell(u)$ (categorical symbols) and node attributes $x(u)$ (may be real vectors)
- Given kernels κ_{label} and κ_{attr} for comparing labels and attributes, the node kernel is

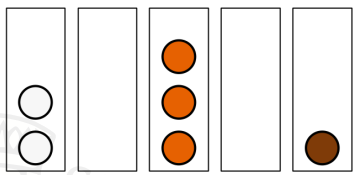
$$\kappa_{\text{node}}(v, v') = \kappa_{\text{label}}(\ell(v), \ell(v'))\kappa_{\text{attr}}(x(v), x(v'))$$

- κ_{label} and κ_{attr} are based on discretization (e.g. via locality-sensitive hashing):

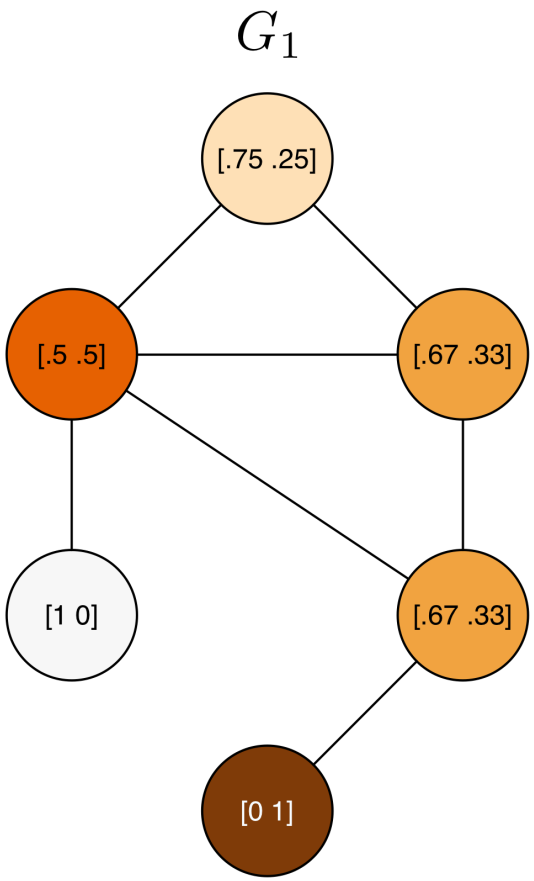
$$\kappa_{\text{label}}(\ell(v), \ell(v')) = \mathbb{1}\{h(p(v)) = h(p(v'))\}$$

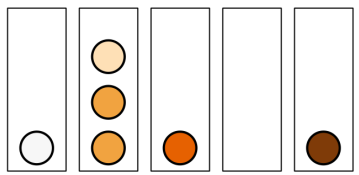
Propagation kernels — Example



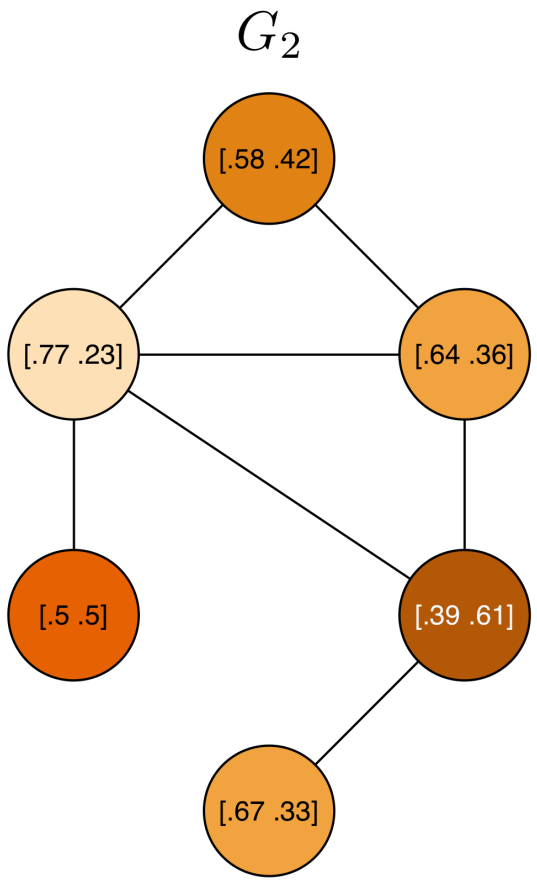


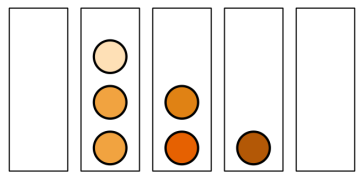
$\phi(G_0) = [2, 0, 3, 0, 1]$





$\phi(G_1) = [1, 3, 1, 0, 1]$





$\phi(G_2) = [0, 3, 2, 1, 0]$

- Start from a given kernel κ_{node} on node attributes (e.g. RBF)

- Start from a given kernel κ_{node} on node attributes (e.g. RBF)
- Define a kernel on paths as follows:
  - Let π_j and π'_j be the vertices at position j in two paths π and π', respectively
  - Let $x(\pi_j)$ and $x(\pi'_j)$ be their (high-dimensional) labels
  - Path kernel:

$$\kappa_{\text{path}}(\pi, \pi') = \mathbb{1}\{|\pi| = |\pi'|\} \sum_{j=1}^{|\pi|} \kappa_{\text{node}}\left(x(\pi_j), x(\pi'_j)\right)$$

- Start from a given kernel κ_{node} on node attributes (e.g. RBF)
- Define a kernel on paths as follows:
  - Let π_j and π'_j be the vertices at position j in two paths π and π', respectively
  - Let $x(\pi_j)$ and $x(\pi'_j)$ be their (high-dimensional) labels
  - Path kernel:

$$\kappa_{\text{path}}(\pi, \pi') = \mathbb{1}\{|\pi| = |\pi'|\} \sum_{j=1}^{|\pi|} \kappa_{\text{node}}\left(x(\pi_j), x(\pi'_j)\right)$$

- Finally define the graph kernel as

$$K(G, G') = \sum_{\pi \in R^{-1}(G)} \sum_{\pi' \in R^{-1}(G')} \kappa_{\text{path}}(\pi, \pi')$$

where $(\pi, G) \in R$ if π is a shortest-path in G

- To speed-up the computation, rewrite the kernel as

$$K(G, G') = \sum_{v \in V} \sum_{v' \in V'} w(v, v') \kappa_{\text{node}}(v, v')$$

where $w(v, v')$ counts the number of times v and v' appear at the same hop in a shortest-path

- To speed-up the computation, rewrite the kernel as

$$K(G, G') = \sum_{v \in V} \sum_{v' \in V'} w(v, v') \kappa_{\text{node}}(v, v')$$

where $w(v, v')$ counts the number of times v and v' appear at the same hop in a shortest-path

- The kernel $w(v, v')$ can be computed as

$$w(v, v') = \langle M(v), M(v') \rangle$$

where $M(v)$ is a $\delta \times \delta$ matrix with entries

$m_{ij}(v) =$ # times v appears at hop i in a shortest-path of length j

and δ is the largest graph diameter

- All matrices $M(v)$ can be computed in $O(V^2(E + \log V + \delta^2))$ calling Dijkstra as a subroutine — see (Feragen et al. 2013) for details

- All matrices $M(v)$ can be computed in $O(V^2(E + \log V + \delta^2))$ calling Dijkstra as a subroutine — see (Feragen et al. 2013) for details

- The overall running time is therefore $O(V^2(d + E + \log V + \delta^2))$ where d is the dimension of the node attribute vector

- All matrices $M(v)$ can be computed in $O(V^2(E + \log V + \delta^2))$ calling Dijkstra as a subroutine — see (Feragen et al. 2013) for details
- The overall running time is therefore $O(V^2(d + E + \log V + \delta^2))$ where d is the dimension of the node attribute vector
- Additionally, $M(v)$ only need to computed once per graph on a given dataset, yielding an amortized running time of $O(dV^2)$
- Nice improvement compared to the running time $O(dV^4)$ of the naive implementation based on the shortest-path kernel

- An invariant is a function $\mathcal{I}$ such that

$$G \approx G' \implies \mathcal{I}(G) = \mathcal{I}(G')$$

- An invariant is a function $\mathcal{I}$ such that

$$G \approx G' \implies \mathcal{I}(G) = \mathcal{I}(G')$$

- The invariant is *complete* if the reverse is also true

- An invariant is a function $\mathcal{I}$ such that

$$G \approx G' \implies \mathcal{I}(G) = \mathcal{I}(G')$$

- The invariant is *complete* if the reverse is also true
- A vertex invariant is a function $\mathcal{L} : V \mapsto \mathcal{C}$ that
  - assigns each vertex v a color $\mathcal{L}(v)$
  - is preserved under any isomorphism f, i.e.

$$\mathcal{L}(v) = \mathcal{L}(f(v))$$

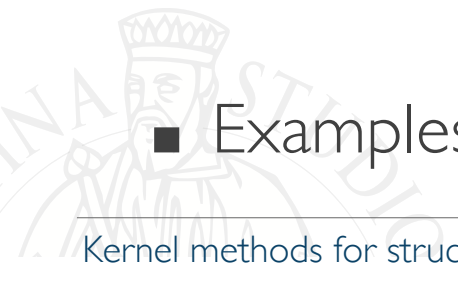
- An invariant is a function $\mathcal{I}$ such that

$$G \approx G' \implies \mathcal{I}(G) = \mathcal{I}(G')$$

- The invariant is *complete* if the reverse is also true
- A vertex invariant is a function $\mathcal{L} : V \mapsto \mathcal{C}$ that
  - assigns each vertex v a color $\mathcal{L}(v)$
  - is preserved under any isomorphism f, i.e.

$$\mathcal{L}(v) = \mathcal{L}(f(v))$$

- Examples: $\mathrm{degree}(v)$, W-L color of v, etc.



Graph invariant kernels (Orsini et al. 2015)

- The key idea to define GIKs is to introduce a notion of structural similarity $w(v, v')$ between two nodes $v \in V$ and $v' \in V'$, based on some invariant

Graph invariant kernels (Orsini et al. 2015)

- The key idea to define GIKs is to introduce a notion of structural similarity $w(v, v')$ between two nodes $v \in V$ and $v' \in V'$, based on some invariant
- Assume a kernel on node attributes κ_{attr} is available

Graph invariant kernels (Orsini et al. 2015)

- The key idea to define GIKs is to introduce a notion of structural similarity $w(v, v')$ between two nodes $v \in V$ and $v' \in V'$, based on some invariant
- Assume a kernel on node attributes κ_{attr} is available
- Define the graph kernel as

$$K(G, G') = \sum_{v \in V} \sum_{v' \in V'} w(v, v') \kappa_{\text{attr}}(x(v), x(v'))$$

i.e. the more two nodes are structurally similar, the more their attribute similarity will contribute to the kernel

Graph invariant kernels (Orsini et al. 2015)

- The key idea to define GIKs is to introduce a notion of structural similarity $w(v, v')$ between two nodes $v \in V$ and $v' \in V'$, based on some invariant
- Assume a kernel on node attributes κ_{attr} is available
- Define the graph kernel as

$$K(G, G') = \sum_{v \in V} \sum_{v' \in V'} w(v, v') \kappa_{\text{attr}}(x(v), x(v'))$$

  i.e. the more two nodes are structurally similar, the more their attribute similarity will contribute to the kernel

- Note that in this setting $x(v)$ may have any type

Graph invariant kernels

- As in many other graph kernels, use a relation R between graphs and their parts: $(g, G) \in R$ iff g is a subgraph of G (i.e. a pattern in G)

Graph invariant kernels

- As in many other graph kernels, use a relation R between graphs and their parts: $(g, G) \in R$ iff g is a subgraph of G (i.e. a pattern in G)
- Furthermore, for a given node v, introduce the relation $R_v \subset R$ such that $(g, G) \in R_v$ iff $(g, G) \in R$ and v is a node in g

- As in many other graph kernels, use a relation R between graphs and their parts: $(g, G) \in R$ iff g is a subgraph of G (i.e. a pattern in G)
- Furthermore, for a given node v, introduce the relation $R_v \subset R$ such that $(g, G) \in R_v$ iff $(g, G) \in R$ and v is a node in g
- Then define the structural similarity between nodes as

$$w(v, v') \doteq \sum_{g \in R_v^{-1}(G)} \sum_{g' \in R_{v'}^{-1}(G')} \kappa_{\text{inv}}(v, v') \frac{\delta(g, g')}{|V_g||V_{g'}|}$$

where $\delta(g, g')$ is used to compare patterns g and g'

- Weisfeiler-Lehman coloring:

$$\kappa_{\text{inv}}(v, v') = \sum_{t=0}^{T} \mathbb{1}\{\mathcal{L}_t(v) = \mathcal{L}_t(v')\}$$

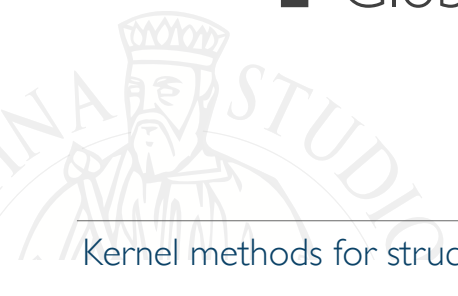
where $\mathcal{L}_t(v)$ is the W-L color of v at iteration t

- Weisfeiler-Lehman coloring:

$$\kappa_{\text{inv}}(v, v') = \sum_{t=0}^{T} \mathbb{1}\{\mathcal{L}_t(v) = \mathcal{L}_t(v')\}$$

  where $\mathcal{L}_t(v)$ is the W-L color of v at iteration t

- Both a local version and a global version of the coloring are possible
  - Local version: run W-L on ech pattern g
  - Global version: run W-L on the whole graph G



- Solve the eigenproblem

$$L\mathbf{x}_i = \lambda_i \mathbf{x}_i$$

where $L = (D - W)$ is the graph Laplacian for a properly choosen weighted adjacency matrix (e.g. use heat kernel)

- Solve the eigenproblem

$$L\mathbf{x}_i = \lambda_i \mathbf{x}_i$$

where $L = (D - W)$ is the graph Laplacian for a properly choosen weighted adjacency matrix (e.g. use heat kernel)

- Define the color vector $\mathcal{L}(v)$ with components

$$\mathcal{L}_i(v) = \begin{cases} |x_i(v)| & \text{if } \lambda_i \text{ has multiplicity } 1 \\ 0 & \text{if } \lambda_i \text{ has multiplicity } > 1 \end{cases}$$

- Solve the eigenproblem

$$L\mathbf{x}_i = \lambda_i \mathbf{x}_i$$

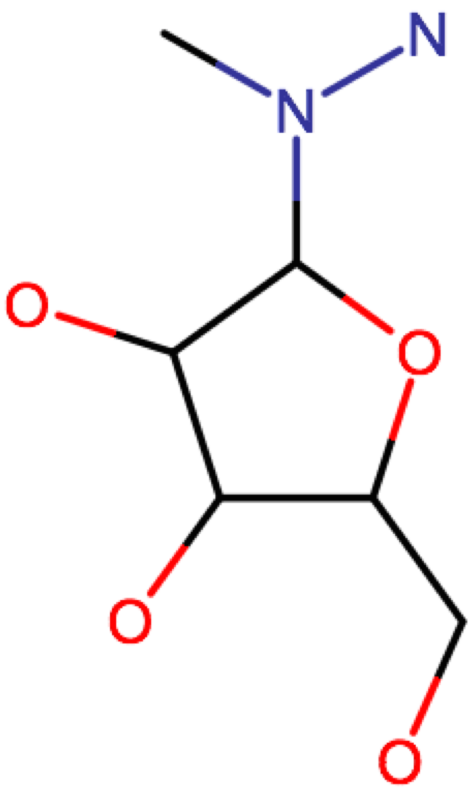
where $L = (D - W)$ is the graph Laplacian for a properly choosen weighted adjacency matrix (e.g. use heat kernel)

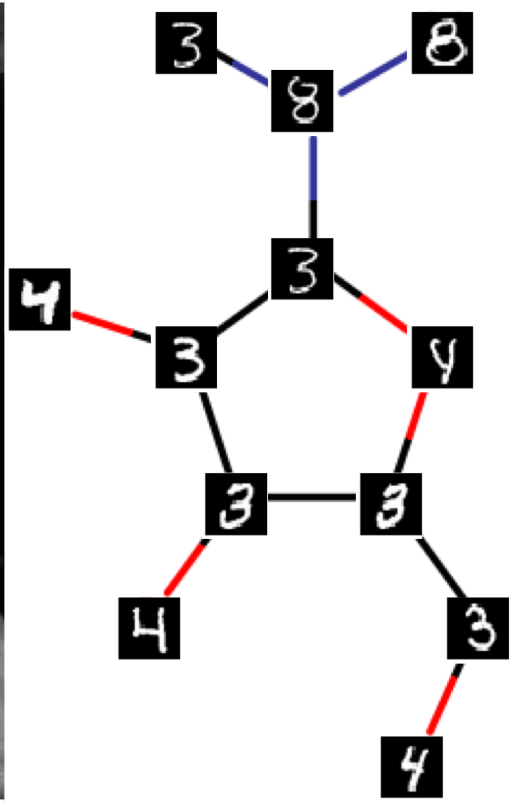
- Define the color vector $\mathcal{L}(v)$ with components

$$\mathcal{L}_i(v) = \begin{cases} |x_i(v)| & \text{if } \lambda_i \text{ has multiplicity } 1 \\ 0 & \text{if } \lambda_i \text{ has multiplicity } > 1 \end{cases}$$

- Let

$$\kappa_{\text{inv}}(v, v') = \exp\left(-\gamma \|\mathcal{L}(v) - \mathcal{L}(v')\|^2\right)$$





- Start from mutagenicity data set of Bursi *et al.*
- Atoms masquerading as MNIST digits

GIK vs GraphHopper

	ENZYMES$_{SYM}$		PROTEIN		SYNTHETIC$_{NEW}$		FRANKENSTEIN		QC	
	WITHOUT CONT.	WITH CONT.	WITHOUT CONT.	WITH CONT.	WITHOUT CONT.	WITH CONT.	WITHOUT CONT.	WITH CONT.	WITHOUT CONT.	WITH CONT.
NSK$_V$	25.9 ± 1.1	71.8 ± 1.0	72.1 ± 0.4	74.2 ± 0.7	78.4 ± 1.9	81.9 ± 1.1	67.9 ± 0.2	72.9 ± 0.3	37.8	92.6
NSK$_{WL}$	56.5 ± 1.1	72.2 ± 0.8	71.7 ± 0.4	76.2 ± 0.4	50.0 ± 0.0	50.0 ± 0.0	74.2 ± 0.3	77.3 ± 0.1	47.8	91.0
GWL$_V$	55.7 ± 1.0	72.6 ± 0.8	**74.9 ± 0.6**	76.1 ± 0.8	**80.8 ± 1.2**	82.8 ± 1.0	73.5 ± 0.3	77.3 ± 0.2	49.8	93.6
GWL$_{WL}$	**58.6 ± 1.4**	71.3 ± 1.1	73.6 ± 0.5	75.8 ± 0.6	50.0 ± 0.0	50.0 ± 0.0	**75.1 ± 0.2**	**78.9 ± 0.3**	48.6	89.6
LWL$_V$	54.5 ± 1.1	**73.3 ± 0.9**	74.4 ± 0.4	**76.6 ± 0.6**	80.6 ± 1.5	**83.0 ± 1.0**	73.0 ± 0.2	77.6 ± 0.2	47.2	**94.6**
LWL$_{WL}$	57.0 ± 1.1	72.0 ± 0.9	71.9 ± 0.6	76.5 ± 0.5	50.0 ± 0.0	50.0 ± 0.0	74.1 ± 0.2	78.3 ± 0.3	47.4	91.8
GSGK$_V$	29.8 ± 0.6	71.8 ± 1.0	73.2 ± 0.3	74.7 ± 0.5	78.2 ± 2.1	82.4 ± 0.9	70.1 ± 0.3	74.0 ± 0.3	44.4	92.6
GSGK$_{WL}$	56.7 ± 1.2	72.2 ± 0.7	72.9 ± 0.5	76.4 ± 0.4	50.0 ± 0.0	50.0 ± 0.0	75.0 ± 0.3	77.6 ± 0.2	47.8	91.0
LSGK$_V$	31.9 ± 1.0	71.9 ± 1.0	72.3 ± 0.4	74.4 ± 0.6	78.7 ± 2.0	82.2 ± 1.1	72.1 ± 0.2	74.9 ± 0.2	42.4	92.2
LSGK$_{WL}$	56.6 ± 1.3	72.1 ± 0.8	71.7 ± 0.3	76.1 ± 0.5	50.0 ± 0.0	50.0 ± 0.0	74.2 ± 0.2	77.4 ± 0.2	**51.4**	91.0
GRAPHHOPPER		69.5 ± 0.7		72.7 ± 0.3		73.9 ± 1.7		68.7 ± 0.4		91.4

Final remarks

- Kernel methods may be effective in relational domains
- Large datasets require $\phi(G)$ but not all available graph kernels allow to compute it explicitly
- Limited by the "fixed-representation" approach: see e.g. (Narayanan et al. 2016; Niepert et al. 2016; Yanardag et al. 2015) for alternatives

References I

Antanas, Laura, McElory Hoffmann, Paolo Frasconi, Tinne Tuytelaars, and Luc De Raedt (2013). "A relational kernel-based approach to scene classification". In: *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*. IEEE, pp. 133–139. url: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6475010`.

Babai, László (2015). "Graph isomorphism in quasipolynomial time". In: *arXiv preprint arXiv:1512.03547*. url: `http://arxiv.org/abs/1512.03547`.

Borgwardt, Karsten M. and Hans-Peter Kriegel (2005a). "Shortest-path kernels on graphs". In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE, 8–pp. url: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1565664`.

Borgwardt, Karsten M. et al. (2005b). "Protein function prediction via graph kernels". In: *Bioinformatics* 21.suppl 1, pp. i47–i56. url: `http://bioinformatics.oxfordjournals.org/content/21/suppl_1/i47.short`.

Collins, Michael and Nigel Duffy (2001). "Convolution kernels for natural language". In: *Advances in neural information processing systems*, pp. 625–632. url: `http://machinelearning.wustl.edu/mlpapers/paper_files/nips02-AA58.pdf`.

References II

Costa, Fabrizio and Kurt De Grave (2010). "Fast neighborhood subgraph pairwise distance kernel". In: *Proceedings of the 26th International Conference on Machine Learning*. Omnipress, pp. 255–262. url: https://lirias.kuleuven.be/handle/123456789/267297.

De Raedt, Luc (2008). *Logical and relational learning*. Springer Science & Business Media. url: https://books.google.it/books?hl=it&lr=&id=FFYIOXvwq7MC&oi=fnd&pg=PA2&dq=Logical+and+relational+learning&ots=nBmYK5moTr&sig=l9CIXBZ7W_SbpK42qRGAZtWUJj8.

Deshpande, Mukund, Michihiro Kuramochi, Nikil Wale, and George Karypis (2005). "Frequent substructure-based approaches for classifying chemical compounds". In: *IEEE Transactions on Knowledge and Data Engineering* 17.8, pp. 1036–1050. url: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1458698.

Feragen, Aasa, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt (2013). "Scalable kernels for graphs with continuous attributes". In: *Advances in Neural Information Processing Systems*, pp. 216–224. url: http://papers.nips.cc/paper/5155-scalable-kernels-for.

Frasconi, Paolo, Fabrizio Costa, Luc De Raedt, and Kurt De Grave (2014). "klog: A language for logical and relational learning with kernels". In: *Artificial Intelligence* 217, pp. 117–143. url: http://www.sciencedirect.com/science/article/pii/S0004370214001064.

References III

Gärtner, Thomas, Peter Flach, and Stefan Wrobel (2003). "On graph kernels: Hardness results and efficient alternatives". In: *Learning Theory and Kernel Machines*. Springer, pp. 129–143. url: http://link.springer.com/chapter/10.1007/978-3-540-45167-9_11.

Haussler, David (1999). *Convolution kernels on discrete structures*. Tech. rep. 646. Department of Computer Science, University of California at Santa Cruz. url: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.411.9684&rep=rep1&type=pdf.

Horváth, Tamás, Thomas Gärtner, and Stefan Wrobel (2004). "Cyclic pattern kernels for predictive graph mining". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 158–167. url: http://dl.acm.org/citation.cfm?id=1014072.

Kashima, Hisashi, Koji Tsuda, and Akihiro Inokuchi (2003). "Marginalized kernels between labeled graphs". In: *ICML*. Vol. 3, pp. 321–328. url: http://www.aaai.org/Papers/ICML/2003/ICML03-044.pdf.

Landwehr, Niels, Andrea Passerini, Luc De Raedt, and Paolo Frasconi (2006). "kFOIL: Learning simple relational kernels". In: *Aaai*. Vol. 6, pp. 389–394. url: http://www.aaai.org/Papers/AAAI/2006/AAAI06-062.pdf.

References IV

Landwehr, Niels, Andrea Passerini, Luc De Raedt, and Paolo Frasconi (2010). "Fast learning of relational kernels". en. In: *Machine Learning* 78.3, pp. 305–342. issn: 0885-6125, 1573-0565. doi: 10.1007/s10994-009-5163-1. url: http://link.springer.com/10.1007/s10994-009-5163-1.

Li, Xin and Dan Roth (2006). "Learning question classifiers: the role of semantic information". In: *Natural Language Engineering* 12.03, pp. 229–249. url: http://journals.cambridge.org/abstract_S1351324905003955.

Menchetti, Sauro, Fabrizio Costa, and Paolo Frasconi (2005). "Weighted decomposition kernels". In: *Proceedings of the 22nd international conference on Machine learning*. ACM, pp. 585–592. url: http://dl.acm.org/citation.cfm?id=1102425.

Narayanan, Annamalai, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan (2016). "subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs". In: San Francisco, CA. url: http://arxiv.org/abs/1606.08928.

Neumann, Marion, Roman Garnett, Christian Bauckhage, and Kristian Kersting (2015). "Propagation kernels: efficient graph kernels from propagated information". en. In: *Machine Learning*, pp. 1–37. issn: 0885-6125, 1573-0565. doi: 10.1007/s10994-015-5517-9. url: http://link.springer.com/article/10.1007/s10994-015-5517-9.

References V

Neumann, Marion, Novi Patricia, Roman Garnett, and Kristian Kersting (2012). "Efficient graph kernels by randomization". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 378–393. url: http://link.springer.com/chapter/10.1007/978-3-642-33460-3_30.

Niepert, Mathias, Mohamed Ahmed, and Konstantin Kutzkov (2016). "Learning Convolutional Neural Networks for Graphs". In: New York, NY, USA. url: http://arxiv.org/abs/1605.05273.

Orsini, Francesco, Paolo Frasconi, and Luc De Raedt (2015). "Graph invariant kernels". In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence. IJCAI*. url: http://ijcai.org/papers15/Papers/IJCAI15-528.pdf.

Schietgat, Leander, Fabrizio Costa, Jan Ramon, and Luc De Raedt (2011). "Effective feature construction by maximum common subgraph sampling". In: *Machine Learning* 83.2, pp. 137–161. url: http://link.springer.com/article/10.1007/s10994-010-5193-8.

Schölkopf, Bernhard and Alexander J. Smola (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press. isbn: 978-0-262-19475-4. url: http://agbs.kyb.tuebingen.mpg.de/lwk/.

Shawe-Taylor, John and Nello Cristianini (2004). *Kernel methods for pattern analysis*. Cambridge university press.

References VI

Shervashidze, Nino, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt (2011). "Weisfeiler-lehman graph kernels". In: *The Journal of Machine Learning Research* 12, pp. 2539–2561. url: `http://dl.acm.org/citation.cfm?id=2078187`.

Shervashidze, Nino, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt (2009). "Efficient graphlet kernels for large graph comparison." In: *AISTATS*. Vol. 5, pp. 488–495. url: `http://www.jmlr.org/proceedings/papers/v5/shervashidze09a/shervashidze09a.pdf`.

Vedaldi, Andrea and Andrew Zisserman (2012). "Efficient additive kernels via explicit feature maps". In: *IEEE transactions on pattern analysis and machine intelligence* 34.3, pp. 480–492. url: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6136519`.

Verbeke, Mathias, Paolo Frasconi, Kurt De Grave, Fabrizio Costa, and Luc De Raedt (2014). "klognlp: Graph kernel–based relational learning of natural language". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, pp. 85–90. url: `https://lirias.kuleuven.be/handle/123456789/451228`.

Verbeke, Mathias et al. (2011). "Kernel-based logical and relational learning with kLog for hedge cue detection". In: *International Conference on Inductive Logic Programming*. Springer, pp. 347–357. url: `http://link.springer.com/chapter/10.1007/978-3-642-31951-8_29`.

References VII

Verbeke, Mathias et al. (2012). "A statistical relational learning approach to identifying evidence based medicine categories". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, pp. 579–589. url: http://dl.acm.org/citation.cfm?id=2391014.

Wale, Nikil, Ian A. Watson, and George Karypis (2008). "Comparison of descriptor spaces for chemical compound retrieval and classification". In: *Knowledge and Information Systems* 14.3, pp. 347–375. url: http://link.springer.com/article/10.1007/s10115-007-0103-5.

Weinberger, Kilian, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg (2009). "Feature Hashing for Large Scale Multitask Learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. New York, NY, USA: ACM, pp. 1113–1120. isbn: 978-1-60558-516-1. doi: 10.1145/1553374.1553516. url: http://doi.acm.org/10.1145/1553374.1553516.

Yanardag, Pinar and S. V. N. Vishwanathan (2015). "Deep graph kernels". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 1365–1374. url: http://dl.acm.org/citation.cfm?id=2783417.