

## Kernels on structures

### Similarity between structured data

- Kernels allow to generalize notion of dot product (i.e. similarity) to arbitrary (non-vector) spaces
- Decomposition kernels suggest a constructive way to build kernels considering *parts* of objects
- Kernels have been developed for the most general structural representations: sequences, trees, graphs.

## Kernels on sequences

### Sequences for data representation

- Variable length objects where order of elements matters
- Biological sequences (DNA, RNA)
- Text documents as sequences of words
- Sequences of sensor readings for human activity

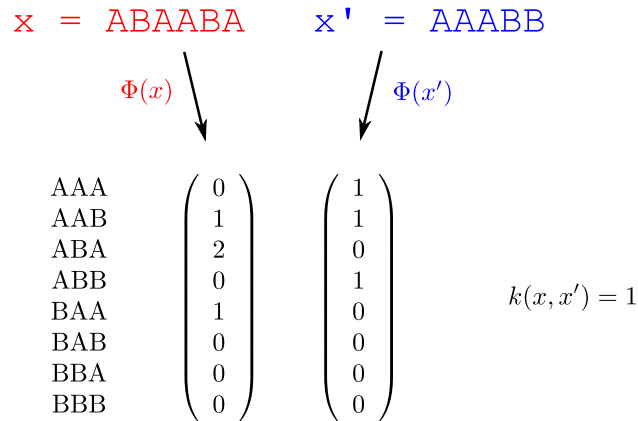
## Kernels on sequences

$$\begin{array}{ccc} x = \text{ABAABA} & & x' = \text{AAABB} \\ \Phi(x) \downarrow & & \downarrow \Phi(x') \\ \begin{array}{l} \text{AAA} \\ \text{AAB} \\ \text{ABA} \\ \text{ABB} \\ \text{BAA} \\ \text{BAB} \\ \text{BBA} \\ \text{BBB} \end{array} & \begin{array}{c} \left( \begin{array}{c} 0 \\ 1 \\ 2 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} \right) & \begin{array}{c} \left( \begin{array}{c} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right) \end{array} & k(x, x') = 1 \end{array}$$

## Spectrum kernel

- Feature space is space of all possible k-grams (subsequences)
- An efficient procedure based on suffix trees allows to compute kernel without explicitly building feature maps

**Kernels on sequences**



**Spectrum kernel: problem**

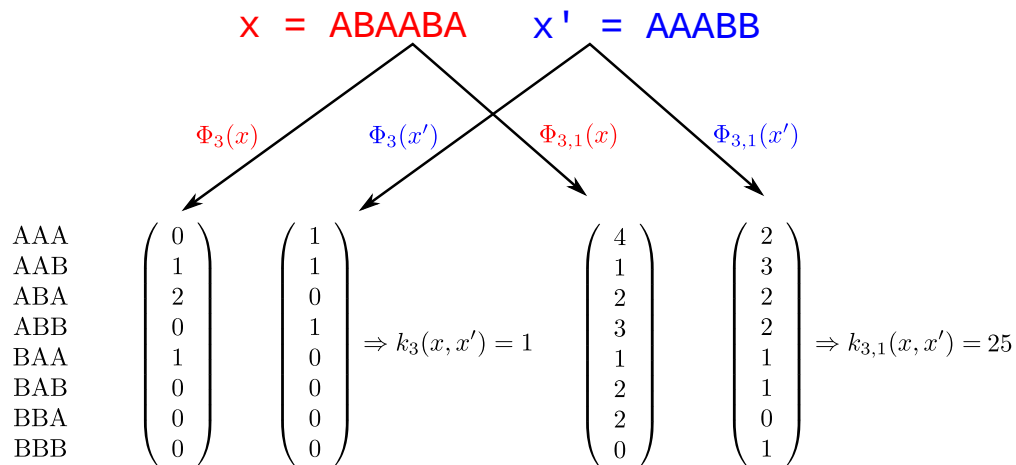
- Feature space representation can be very sparse (many zero features, especially for high  $k$ )
- Sparse feature maps tend to produce orthogonal examples (an example is only similar to itself)

**Kernels on sequences**

**Mismatch string kernel**

- Allows for approximate matches between  $k$ -grams
- Defines a  $(k-m)$ -neighbourhood of a  $k$ -gram as all  $k$ -grams with at most  $m$  mismatches to it
- Each  $k$ -gram counts as a feature for its entire  $(k-m)$ -neighbourhood
- The kernel can be efficiently computed using a  $(k-m)$ -mismatch tree (similar to suffix tree)

**Kernels on sequences**



**Mismatch string kernel**

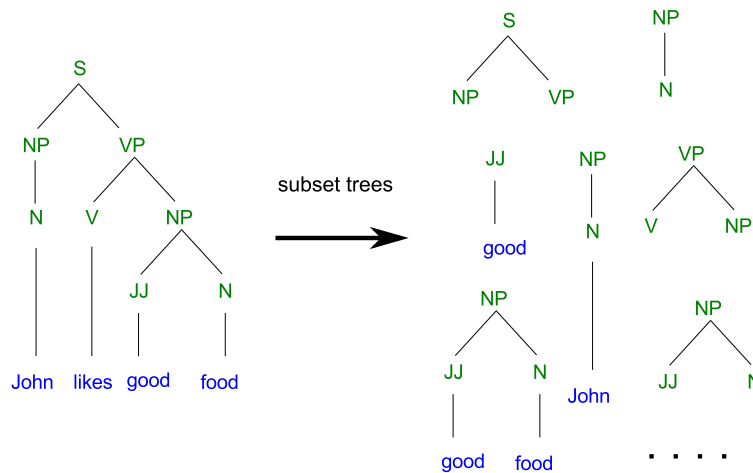
- The feature map is denser than that of the spectrum kernel

## Kernels on trees

### Trees for data representation

- Objects having hierarchical internal representation
- Taxonomies of concepts in a domain
- E.g. phylogenetic trees representing evolution of organisms
- Parse trees representing syntactic structure of sentences

### Kernels on trees



### Subset tree kernel

- A subset tree is a subtree having either all or no children of a node (and is not a single node)
- A subset tree kernel corresponds to a feature map of all subset trees
- It is a special type of tree-fragment kernel (many other exist), justified by grammatical considerations (do not break a grammar rule)

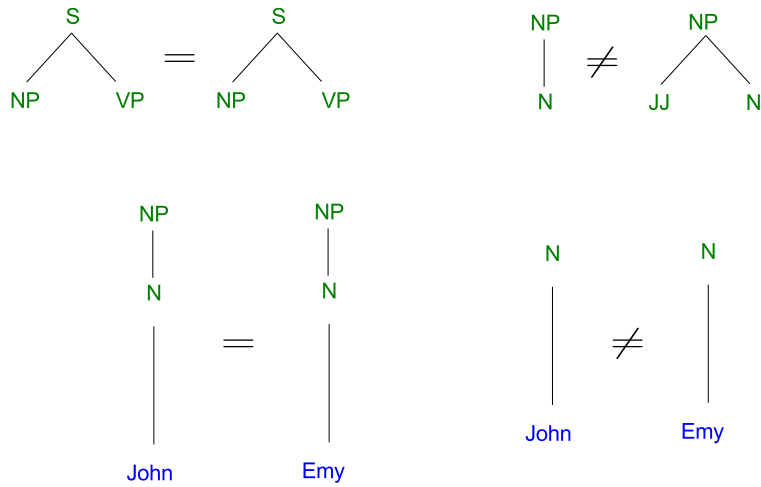
## Kernels on trees

### Subset tree kernel

$$k(t, t') = \sum_{i=1}^M \phi_i(t) \phi_i(t') = \sum_{n_i \in t} \sum_{n'_j \in t'} C(n_i, n'_j)$$

- The subset tree kernel is the product of the subset tree mapping  $\Phi(\cdot)$  of the two trees  $t$  and  $t'$ .
- It can be computed summing the number of common subtrees  $C(n_i, n'_j)$  rooted at nodes  $n_i, n'_j$ , for all  $n_i$  and  $n'_j$

**Kernels on trees**



**Subset tree: node matching**

- Two nodes  $n_i, n'_j$  match if:
  1. they have the same label
  2. they have the same number of children
  3. each child of  $n_i$  has the same label of the corresponding child of  $n'_j$

**Kernels on trees**

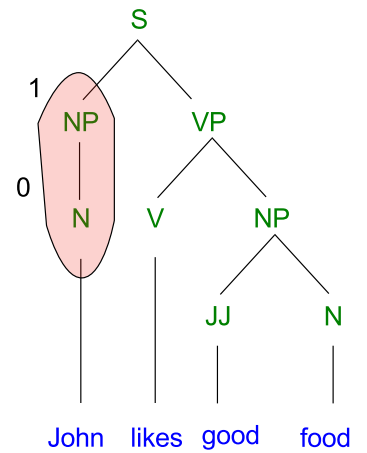
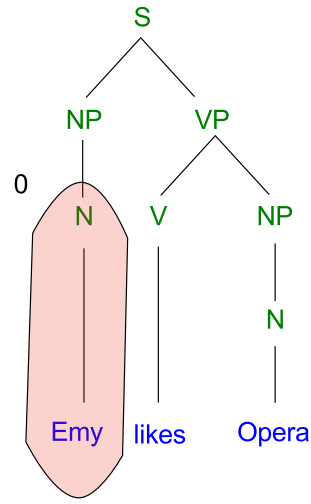
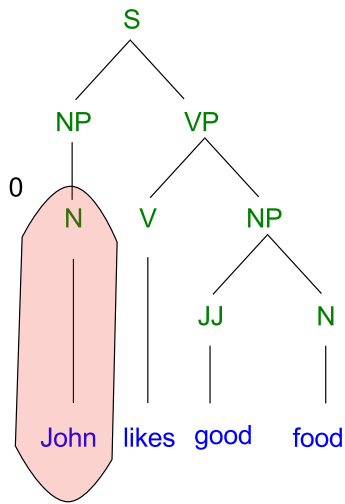
**Recursive procedure for  $C(n_i, n'_j)$**

- If  $n_i$  and  $n'_j$  don't match  $C(n_i, n'_j) = 0$ .
- if  $n_i$  and  $n'_j$  match, and they are both pre-terminals (parents of leaves)  $C(n_i, n'_j) = 1$ .
- Else

$$C(n_i, n'_j) = \prod_{j=1}^{nc(n_i)} (1 + C(ch(n_i, j), ch(n'_j, j)))$$

where  $nc(n_i)$  is the number of children of  $n_i$  (equal to that of  $n'_j$  for the definition of match) and  $ch(n_i, j)$  is the  $j^{th}$  child of  $n_i$ .

**Kernels on trees**



## Kernels on trees

### Dominant diagonal

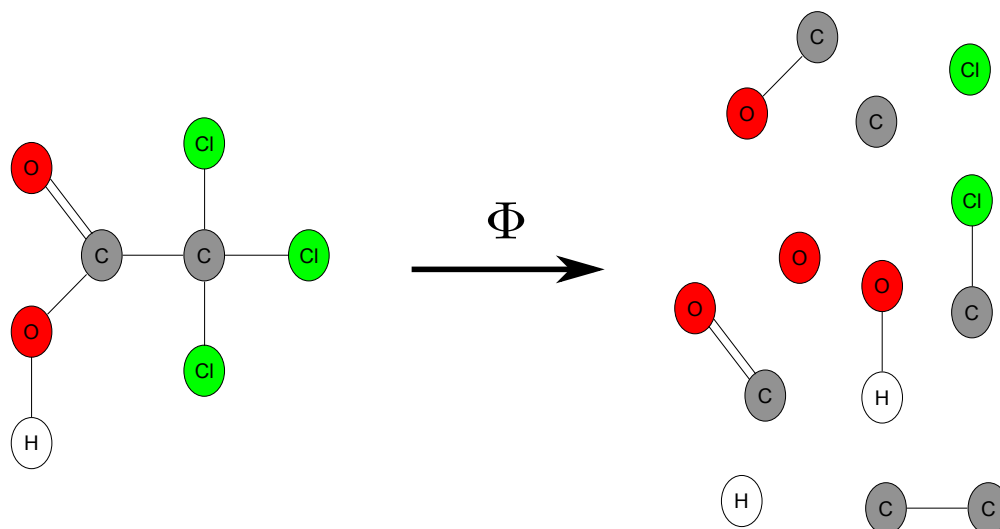
- The kernel value strongly depends on the size of the tree (normalize!!)
- It is difficult that very large portion of trees are identical in different examples
- Similarity of example to itself tend to be orders of magnitude higher than to any other example (*dominant diagonal* problem)
- One solution consists of downweighting larger subtrees:
  - simply replace 1 by  $0 \leq \lambda \leq 1$  in previous procedure

## Kernels on graphs

### Graphs for data representation

- graphs are a powerful formalism allowing to represent data with arbitrary structures
- Chemical molecules are commonly represented as graphs made of atoms and bonds
- Networked data (e.g. a web site, the Internet) can be naturally encoded as graphs

## Kernels on graphs



## Bag of subgraphs

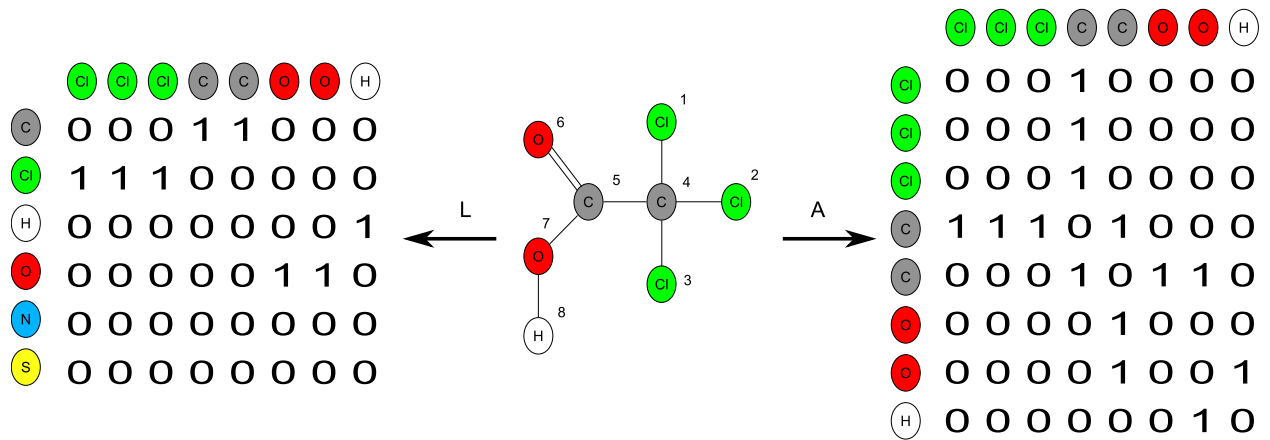
- One feature for all possible subgraphs up to a certain size (2 in figure)
- Feature value is frequency of occurrence of subgraph
- PB of graph isomorphisms (ok for small subgraphs)

## Kernels on graphs

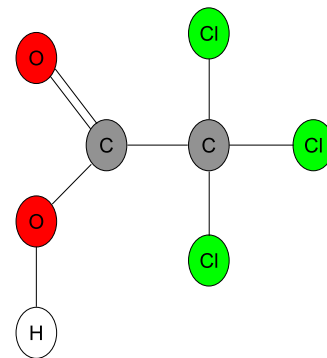
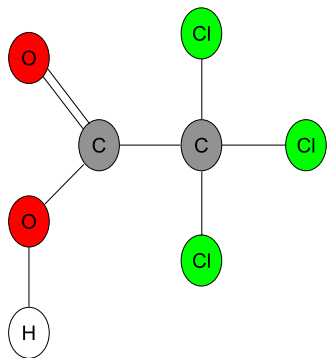
### Main definitions

- A graph  $G = (\mathcal{V}, \mathcal{E})$  is a finite set of vertices (or nodes)  $\mathcal{V}$  and edges  $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$
- A (node)labelled graph is a graph whose nodes are labelled with symbols  $label(v_j) = \ell_i$  from  $\mathcal{L}$ .
- A (node)labelled graph can be also encoded with:
  - A square *adjacency* matrix  $A$  such that  $A_{ij} = 1$  if  $(v_i, v_j) \in \mathcal{E}$  and 0 otherwise
  - A (node)label matrix  $L$  such that  $L_{ij} = 1$  if  $label(v_j) = \ell_i$  and zero otherwise

## Kernels on graphs: definitions



**Kernels on graphs**



**Walk kernels**

- A walk in a graph is a sequence of nodes  $\{v_1, \dots, v_{n+1}\}$  such that  $(v_i, v_{i+1}) \in \mathcal{E}$  for all  $i$
- The length of a walk is the number of its edges
- The set of all walks of length  $n$  is written as  $W_n(G)$

**Kernels on graphs**

**Walk kernels**

- A possible walk kernels compares graphs considering the set of walks starting and ending with the same labels  $\ell_{start}, \ell_{end}$ .

- This corresponds to having a feature for all possible label pairs  $\ell_i, \ell_j$  with value:

$$\phi_{\ell_i, \ell_j}(G) = \sum_{n=1}^{\infty} \lambda_n |\{(v_1, \dots, v_{n+1}) \in W_n(G) : l(v_1) = \ell_i \wedge l(v_{n+1}) = \ell_j\}|$$

- i.e. a weighted (by  $\lambda_n \geq 0$  for all  $n$ ) sum of the number of walks starting with label  $\ell_i$  and ending with label  $\ell_j$

## Kernels on graphs

### Walk kernels

- The  $n^{\text{th}}$  power of the adjacency matrix,  $A^n$ , computes the number of walks of length  $n$  between any two nodes.
- I.e.  $(A^n)_{ij}$  is the number of walks of length  $n$  between  $v_i$  and  $v_j$
- This can be used to efficiently compute the overall feature map as:

$$\phi_{\ell_i, \ell_j}(G) = \left( \sum_{n=1}^{\infty} \lambda_n L A^n L^T \right)_{\ell_i, \ell_j}$$

## Kernels on graphs

### Walk kernels

- The corresponding kernel is:

$$k(G, G') = \left\langle L \left( \sum_{i=1}^{\infty} \lambda_i A^i \right) L^T, L' \left( \sum_{j=1}^{\infty} \lambda_j A'^j \right) L'^T \right\rangle$$

where the dot product between two matrices  $M, M'$  is defined as:

$$\langle M, M' \rangle = \sum_{i,j} M_{ij} M'_{ij}.$$

### Exponential graph kernel

- An example of walk kernel is:

$$k_{exp}(G, G') = \langle L e^{\beta A} L^T, L' e^{\beta A'} L'^T \rangle$$

where  $\beta \in \mathbb{R}$  is a parameter

## References

**string kernels** J.Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004 (Section 9)

**tree kernels** M. Collins and N. Duffy. *Convolution kernels for natural language*. In , Advances in Neural Information Processing Systems 14, Cambridge, MA, 2002. MIT Press.

**graph kernels** Thomas Gärtner. *Exponential and Geometric Kernels for Graphs*. NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data, 2002