Kernels for Structured Data

Davide Bacciu

Dipartimento di Informatica
Università di Pisa
bacciu@di.unipi.it

Machine Learning: Neural Networks and Advanced Models (AA2)

Today's Lecture

- A refresher on kernel methods
- Kernel methods for structured data
  - Sequences, trees, graphs
- Design guidelines for kernels
  - Convolutional kernels
  - Adaptive kernels
  - Generative kernels
- Experimental analysis

Intuition and Motivations

- Kernels can be interpreted as similarity functions $k(x_1, x_2)$ of their two arguments (data points x_1 and x_2)
- Use the kernel function within a known classifier/regressor to
  - Extend linear learning models to non-linear approaches
    - Support Vector Machines
    - Kernel PCA,...
  - Extend learning models to new classes of data
    - Sequences, trees, ...

Kernel Functions

- A kernel is defined by a scalar product

$$k(x_1, x_2) = \Phi(x_1)^T \Phi(x_2)$$

  - $\Phi(x) : \mathcal{D} \to \mathcal{F}$ is a fixed nonlinear mapping from data space $\mathcal{D}$ to feature space $\mathcal{F}$
- Given a dataset $\{x_1, \ldots, x_N\}$ define the Gram matrix

$$\mathbf{K} = \begin{bmatrix} K(x_1, x_1) & \ldots & K(x_1, x_N) \\ \ldots & \ldots & \ldots \\ K(x_N, x_1) & \ldots & K(x_N, x_N) \end{bmatrix}$$

- A function $k(x_1, x_2)$ is a kernel if its Gram matrix is positive semidefinite

$$\forall v \in \mathbb{R}^N, v^T \mathbf{K} v \geq 0$$

The nonlinear feature mapping $\Phi(x)$ does not need to be known

Kernel Trick and Feature Space

- Take an known algorithm defined in terms of scalar products between data points and substitute them with the kernel $k(x_1, x_2)$
- Can be performed using any kernel function and results in a kernelization of the algorithm
  - E.g. kernel PCA, kernel k-means, ...
- It amount to solving the learning problem in the feature space $\mathcal{F}_\Phi$ induced by the non-linear map $\Phi(x)$, without requiring to know the form of Φ

How to Construct a Kernel?

- Define a function $k(x_1, x_2)$ measuring some form of similarity between data points
  - Prove it is positive semi-definite
  - Use the kernel-trick to express vector products using the kernel function only
- If the feature space $\mathcal{F}$ is known, try defining a fast way to compute the inner product $\Phi(x_1)^T \Phi(x_2)$
  - Eg. string, tree and graph kernels
- Combine existing kernels
  - Weighted sum, product and tensor product of kernels
  - Concatenation, exponentiated kernel, ...

Examples of Kernels for Vectorial Data

Polinomial of degree up to d

$$k(x_1, x_2) = (x_1^T x_2 + c)^d$$

Exponential kernel (infinite-dimensional feature space)

$$k(x_1, x_2) = \exp\left(s \cdot (x_1^T x_2)\right)$$

Gaussian kernel

$$k(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

Kernels for Non-Vectorial Data

The definition of kernel (similarity) functions for non-vectorial data allows to straightforwardly extend well-known algorithms to new classes of data using the kernel trick

- Data: sequences, tree, graphs, distributions, structured spaces
- Learning Tasks: classification, clustering, visualization,...
- Applications: molecule function prediction, vision, DNA sequences classification

Focus on kernels for structured data

- Often use explicit formulation of the feature space
- Counting (and weighting) matching substructures
- Some adaptive approaches (learning kernel from data)

Kernel Types

Different types of kernels for structured data based on adaptivity and compositionality properties

- Convolutional kernels
  - Decompose structured objects into parts whose similarity can be measured
  - Aggregate the similarities measured on parts to compute the structure match
- Syntactic kernels
  - Convolutional kernels counting the number of common substructures in the objects
  - Weight matching node labels, edges, paths,...
- Adaptive kernels
  - Learn the weight of a structure-substructure match from data
  - Data population induces the similarity metric

Generative Kernels

- A combination of discriminative and generative models
  - Use a generative model to define a kernel
  - Use the kernel in a discriminative approach (e.g. SVM classification)
- Syntactic kernels
  - Use a probabilistic model to generate substructures to be visited in convolutional kernels
  - E.g. marginalized graph kernel, ...
- Adaptive kernels
  - Fit a probabilistic model to the (structured) data
  - Use the properties of the fitted distribution to measure object similarity
  - E.g. Fisher kernel, Jaccard generative kernel, ...

String Matching Kernels

A convolutional approach to measure similarity $K(\mathbf{x}_1, \mathbf{x}_2)$ between strings $\mathbf{x}_1$ and $\mathbf{x}_2$

- Count the common substrings within $\mathbf{x}_1$ and $\mathbf{x}_2$
  - A matching substring is given weight 1
  - Non-matching substrings weight 0
  - Overall kernel is given by the sum on all substrings
- Explicit approach
  - Feature space encoding $\Phi_s(\mathbf{x}) =$ number of occurrences of string s in $\mathbf{x}$
  - Compute kernel as $K(\mathbf{x}_1, \mathbf{x}_2) = \Phi_s(\mathbf{x}_1)^T \Phi_s(\mathbf{x}_2)$
- Implicit approach
  - Count substrings of $\mathbf{x}_1$ (i.e. $\mathcal{S}(\mathbf{x}_1)$) occurring in $\mathbf{x}_2$

$$K(\mathbf{x}_1, \mathbf{x}_2) = \sum_{s_1 \in \mathcal{S}(\mathbf{x}_1)} \sum_{s_2 \in \mathcal{S}(\mathbf{x}_2)} \mathbb{I}(s_1, s_2)$$

  - Computationally more efficient ($O(|\mathbf{x}_1| \cdot |\mathbf{x}_2|)$) as it does not need to explore all strings in vocabulary

Generative String Kernels

- Use a generative model for sequences to obtain an adaptive measure of string similarity
- First, need a probability distribution $P(\mathbf{x})$ over sequences $\mathbf{x}$
  - Fit an Hidden Markov Model to the training data
- Compute the kernel using the information in $P(\mathbf{x})$
  - Two sequences are similar if both have high probability

$$K(\mathbf{x}_1, \mathbf{x}_2) = P(\mathbf{x}_1)^T P(\mathbf{x}_2)$$

  - Two sequences are similar if are generated by same hidden states

$$K(\mathbf{x}_1, \mathbf{x}_2) = \sum_{\mathbf{z}} P(\mathbf{x}_1|\mathbf{z}) P(\mathbf{x}_2|\mathbf{z}) P(\mathbf{z})$$

The Fisher Kernel

- A general approach to obtain a kernel from any generative model $P(\mathbf{x}|\theta)$ parameterized by θ
  - With sequential data we consider an HMM with parameters $\theta = \{A, B, \pi\}$
- The feature space encoding of $\mathbf{x}$ is the Fisher score

$$\Phi(\mathbf{x}) = \Delta_\theta \log P(\mathbf{x}|\theta)$$

  - Represent the contribution of each model parameter to input sample generation
- The practical Fisher kernel is simply

$$K(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1)^T \Phi(\mathbf{x}_2)$$

- Computational complexity is $O(|\theta|)$

Fisher String Kernel in Practice

- The Fisher score for sequences is obtained by differentiating the HMM log-likelihood w.r.t. the parameters $\theta = \{A, B, \pi\}$
- The process is similar to that performed to obtain the EM learning equations
  - Requires to perform a forward-backward recursion
  - $O(|\mathbf{X}| \cdot |\theta|)$
- If you do the math, the Fisher score is basically the ratio between the posterior and each model parameters

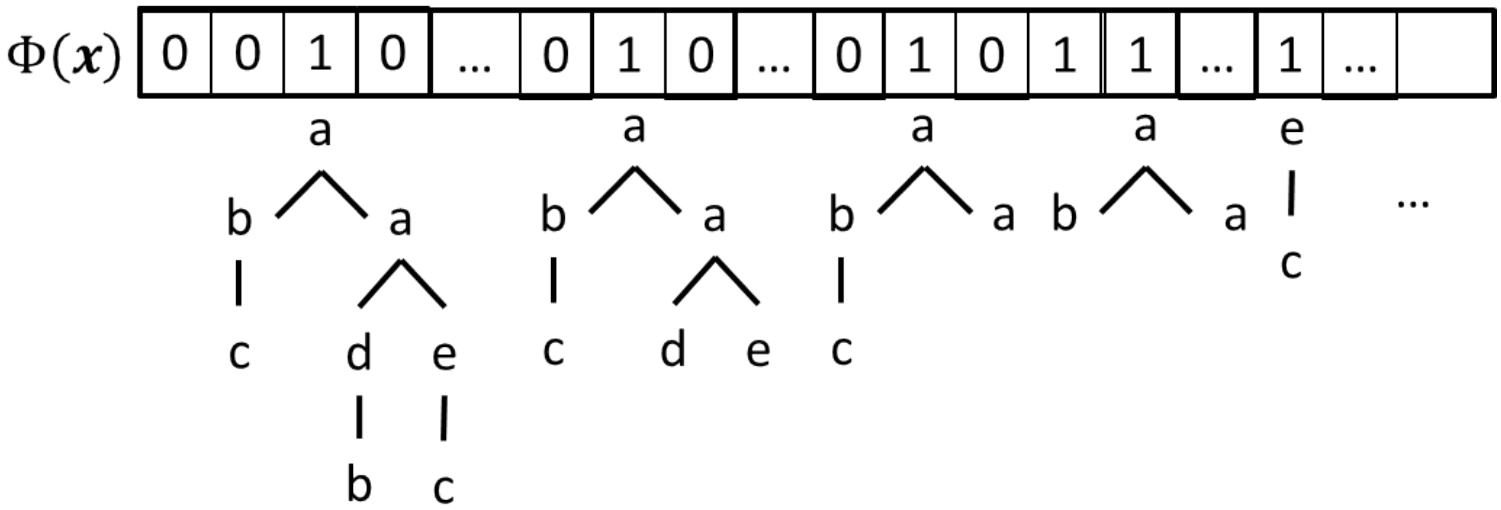
The Fisher kernel is generative and adaptive but it is not convolutional

Convolutional Tree Kernels

- Find syntactic matches on the tree substructures
  - Count number of common paths between trees, e.g. using a string kernel
  - Count number of matching subtrees between trees
- Several convolutional tree kernels that consider different subsets of subtrees
  - Different expressiveness, i.e. capability to capture structural matches
  - Different computational complexity
- Subset tree (SST) kernel ($O(|\mathbf{x}|^2)$)
  - Count the number of matching proper subtrees between two input trees
- Subtree (ST) kernel ($O(|\mathbf{x}| \log |\mathbf{x}|)$)
  - Restrict to matching only complete subtrees (only descendants of subtree root until leaves)

Subset Tree (SST) kernel

Explicit formulation



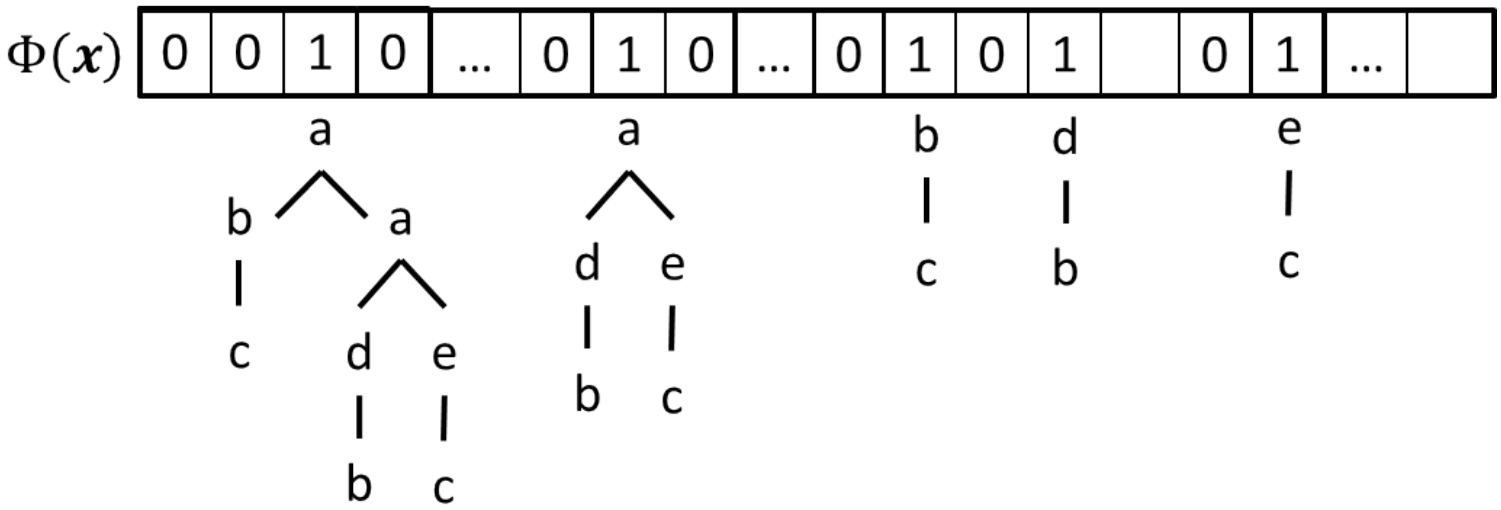
Implicit formulation (recursive)

- If $|Ch(\mathbf{x}^1)| \neq |Ch(\mathbf{x}^2)|$, $K(\mathbf{x}^1, \mathbf{x}^2) = 0$
- Else if x^1 and x^2 are leaves and $x^1 = x^2$, $K(x^1, x^2) = 1$
- Else

$$K(\mathbf{x}^1, \mathbf{x}^2) = \prod_{u=1}^{|Ch(\mathbf{x}^1)|} (1 + K(\mathbf{x}_u^1, \mathbf{x}_u^2))$$

Subtree (ST) kernel

Explicit formulation



Implicit formulation (recursive)

- If $|Ch(\mathbf{x}^1)| \neq |Ch(\mathbf{x}^2)|$, $K(\mathbf{x}^1, \mathbf{x}^2) = 0$
- Else if x^1 and x^2 are leaves and $x^1 = x^2$, $K(x^1, x^2) = 1$
- Else

$$K(\mathbf{x}^1, \mathbf{x}^2) = \prod_{u=1}^{|Ch(\mathbf{x}^1)|} (K(\mathbf{x}_u^1, \mathbf{x}_u^2))$$

Generative Tree Kernels

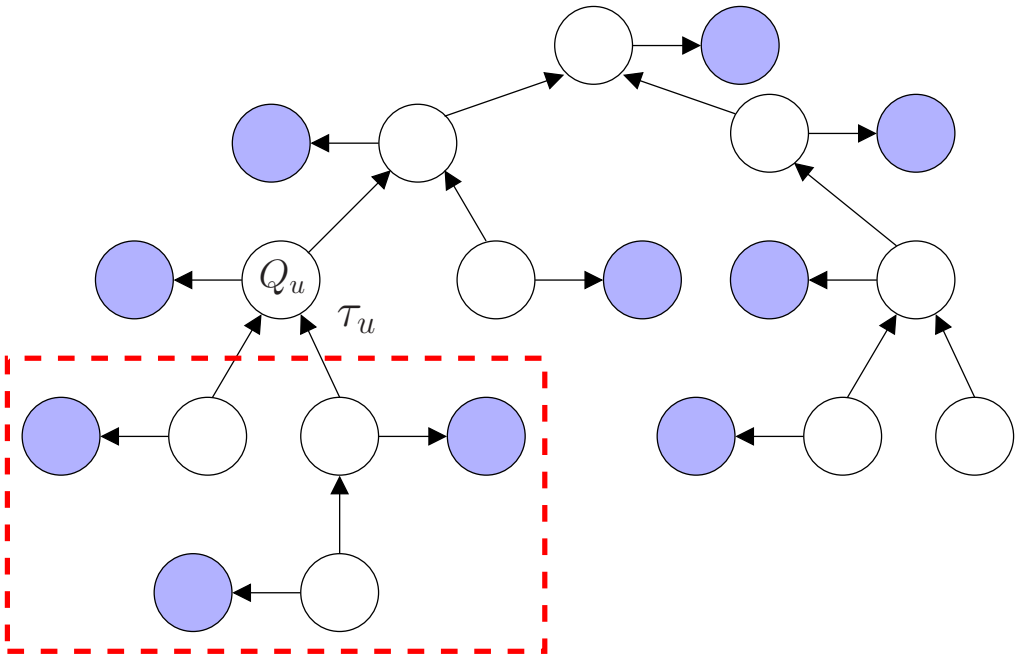
Hidden Tree Markov Models (HTMM)

Define a probability distribution over trees $P(\mathbf{x}|\theta)$ regulated by hidden state variables Q_u (Top-down Vs Bottom-up generation)

- Exploit the information in HTMM to define adaptive generative kernels for trees
- Fisher kernel approach ($O(|\theta|)$)
  - Derive the Fisher score vector for the HTMM parameters θ
  - Can be computed from the upwards-downwards algorithm
- Hidden states multiset kernel ($O(C^2)$)
  - Find a compact feature space encoding the information captured by the HTMM hidden states
  - Use Jaccard similarity to compute the kernel from the encoding

Bottom-up (BU) Tree Context

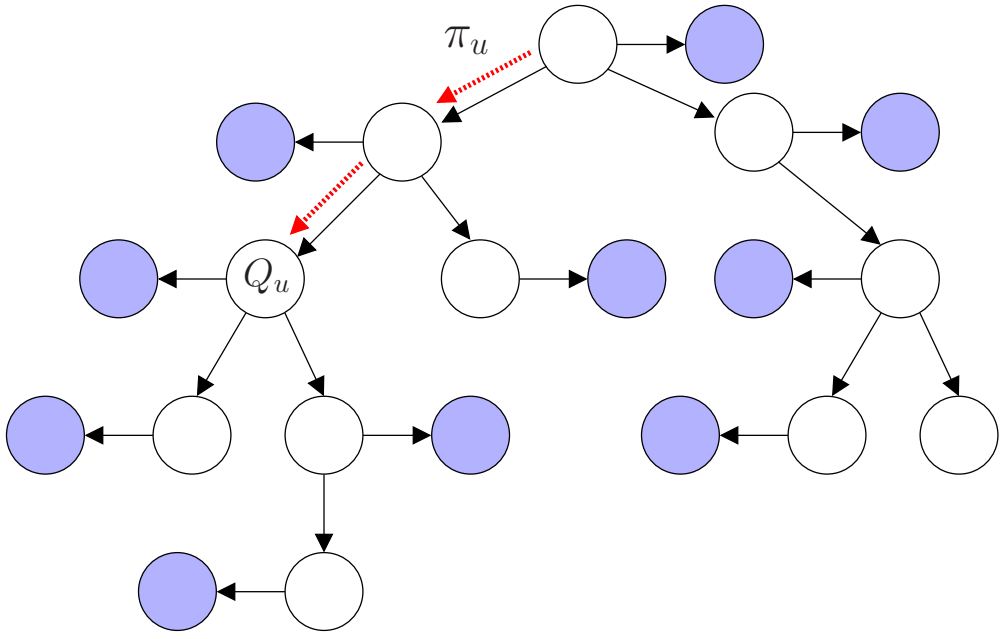
Hidden state Q_u summarizes information concerning structural properties of subtree τ_u rooted in u



BU hidden state space provides a summarized view of the subtrees occurring in the data, where each hidden state identifies a cluster of similar structures

Top-down (TD) Tree Context

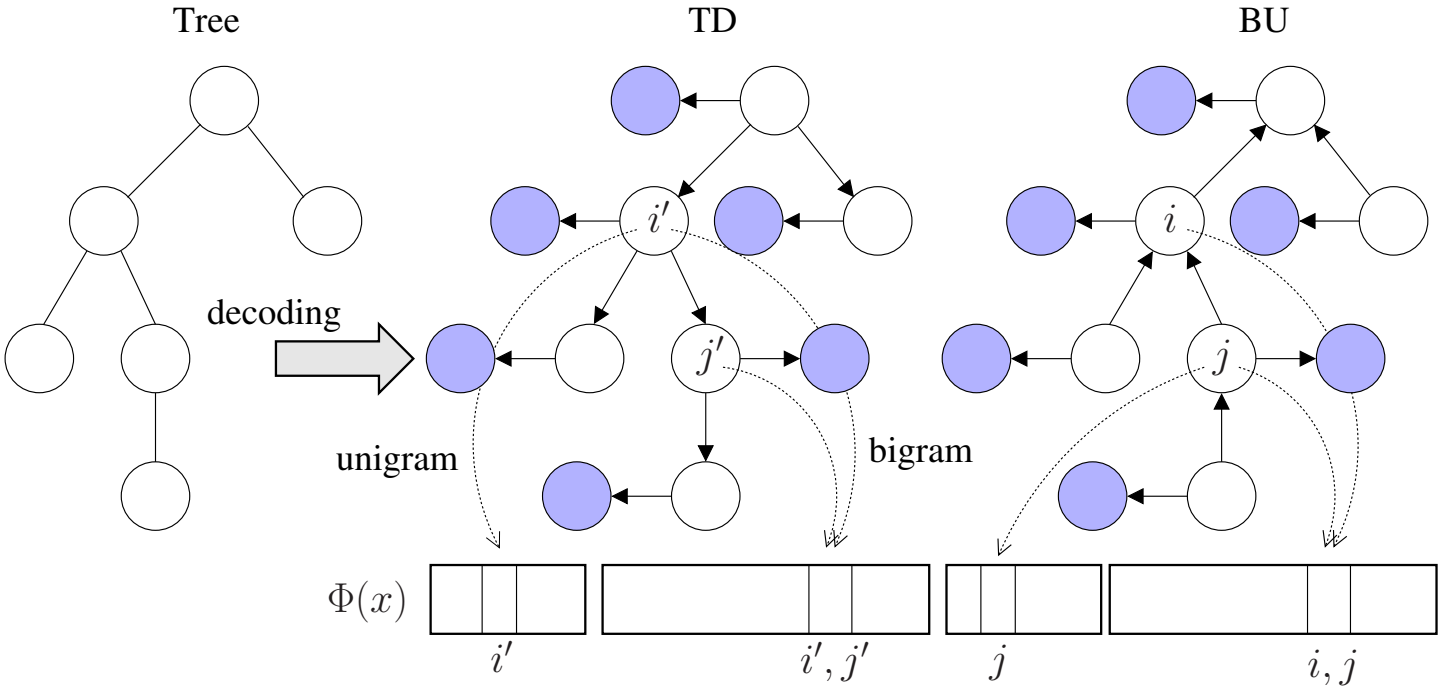
Hidden state Q_u captures information about path π_u leading to the node from the root



TD hidden state space provides a summarized view where each hidden state clusters similar root-to-node paths

Hidden States Multisets

A tree $\mathbf{x}_n$ is transformed into a vector $\Phi(\mathbf{x}_n)$ of hidden states counts from TD and BU models



Compute the Jaccard kernel as

$$k(\mathbf{x}_1, \mathbf{x}_2) = \frac{\sum_{i=1}^{D} \min(\Phi_i(\mathbf{x}_1), \Phi_i(\mathbf{x}_2))}{\sum_{i=1}^{D} \max(\Phi_i(\mathbf{x}_1), \Phi_i(\mathbf{x}_2))}$$

Tree Classification Example

XML document classification benchmarks from the INEX 2005 and 2006 competitions

Table: Test accuracy (%) on models selected by 3-fold CV, using C-SVM classifier in LibSVM

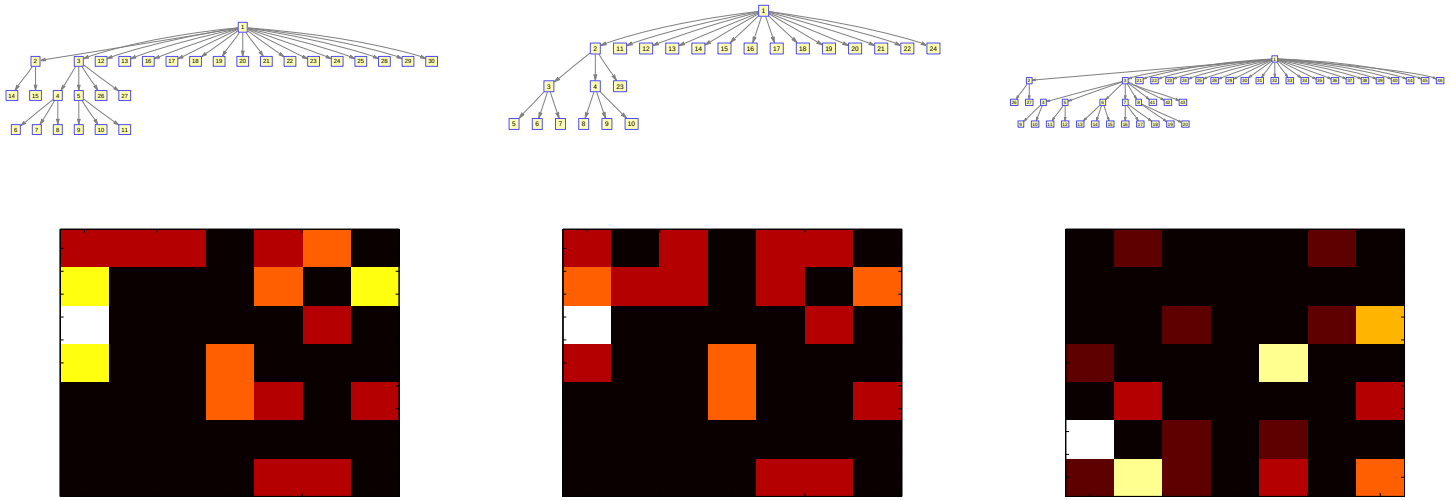
Dataset	Fisher	Jac-BU	Jac-TD	Jac-TB
INEX 2005	96.82 (0.1)	94.22 (0.81)	93.39 (2.19)	95.39 (0.14)
INEX 2006	39.47 (0.8)	44.53 (0.09)	44.38 (0.06)	44.78 (0.02)

Table: Test accuracy (%) by syntactic kernels

Dataset	ST	SST	Poly-SST
INEX 2005	88.73	88.79	88.33
INEX 2006	32.02	40.41	40.12

Activation Masks (AM)

- Topographic maps (e.g. GMT-SD) naturally encode information on tree similarity
  - Structures and substructures are projected on points of the map
  - Similar structures tend to end-up close on the map



Can we devise a kernel for GTM-SD that exploits this intuition?

AM Kernel on GTM-SD

- Given two trees $\mathbf{x}^1$ and $\mathbf{x}^2$ obtain the projection of their nodes the map, e.g. c_u and $c_{u'}$
- Compute the AM Kernel (adaptive, generative, convolutional)

$$k(\mathbf{x}^1, \mathbf{x}^2) = \sum_{u \in \mathcal{U}_1} \sum_{u' \in \mathcal{U}_2} T_\epsilon(c_u, c_{u'})$$

using the weight function

$$T_\epsilon(c_u, c_{u'}) = \begin{cases} \epsilon - d(c_u, c_{u'}), & \text{if } d(c_u, c_{u'}) \leq \epsilon \\ 0, & \text{otherwise} \end{cases}$$

Size	μGTM-SD test error	AM-GTM test error		
		$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.2$
20×20	7.52	3.3673	**3.3465**	3.4296
15×15	9.12	3.881	**3.4712**	3.4505
10×10	7.21	3.5130	**3.4089**	3.6535
9×9	13.13	**3.4504**	3.3049	3.3673

Comparing Graphs - The Isomorphism Problem

Graph Isomorphism

Find a mapping between vertices of graphs G and H such the graphs are identical

- Unknown polynomial-time algorithm
- No reduction to NP complete problems

Subgraph Isomorphism

Find if a subset of vertices and edges of G can be made isomorphic to a subset of H

- Known to be NP complete

A Quick View on Graph Kernels

- Design kernels that compare substructures of graphs that are computable in polynomial time
  - Walks, paths, trees, cyclic patterns,...
  - Expressive, efficient, positive definite, general
- A convolutional approach
  1. Generate a number of graph visits to obtain target substructures
  2. Use a syntactic kernel to match substructures in a convolutional way
- Marginalized kernels
  - A family of generative kernels using a probabilistic approach to generate graph visits
  - Not an adaptive approach!

Random Walks Kernel

- Compare walks in two input graphs
- Walks are node sequences allowing node repetitions
- Computational tricks
  - Build product graph consisting of pairs of identically labeled nodes and edges in 2 graphs
  - Use the powers of the product graph adjacency matrix to check paths of length k
  - Define a kernel counting pairs of matching walks
- Complexity is $O(N^6)$-$O(N^3)$
- Tottering - Walks may visit same edges and nodes multiple times yielding to artificially high similarity scores

Random Trees Kernel

- Compare tree-like substructures of graphs
- May distinguish between substructures that walk kernel deems identical
- Key idea
  - For all pair of nodes in the two graphs construct subtrees of bounded depth h
  - Use a tree kernel to compute match with a convolutional approach
- Computational complexity influenced by tree kernel
- Still affected by tottering

Take Home Messages

- Kernel methods provide a powerful and straightforward way to
  - Extend the classes of data to which learning models can be applied: structured data
  - Allow linear approaches to deal with non-linear problems (next lecture)
- Kernels for structured data
  - Feature space often explicit
  - Implicit formulation might be computationally more convenient
- Generative kernels
  - A general approach to define kernels where matching weights are inferred from data
  - Exploit the expressiveness of generative models with the discriminative power of kernels
  - May result in very efficient kernels