Kernel methods in machine learning lecture 9

Kernel methods with structured and multi-view data

Riikka Huusari
Postdoctoral researcher

Aalto University
Kernel Methods, Pattern Analysis and Computational Metabolomics (KEPACO)

riikka.huusari@aalto.fi

5.4.2021

Overview

Overview

Why kernel methods when there is deep learning?

No universal solution to everything; these methods work for different problems.

Different complexities:

- Kernel methods depend on the $n \times n$ kernel matrix; algorithms often $\mathcal{O}(n^3)$. Need to learn $\mathcal{O}(n)$ parameters.
  $\Rightarrow$ can tackle high-dimensional and complex features well, can easily tackle medium-size datasets $n \approx 10^4$
- Deep networks generally scale $\mathcal{O}(nd)$ with d the input dimension. Need to learn $\mathcal{O}(d)$ parameters[1]
  $\Rightarrow$ can tackle a lot of data, but the dimensionality is a restricting factor if it gets very high.

[1] for dense neural networks

Why kernel methods when there is deep learning?

Deep networks incorporate less prior knowledge, and use (and overfit) more data

Big data vs small data

- Automated data gathering from for example social networks; huge datasets ⇒ neural networks
- Expensive experimentations done by humans or with expensive machinery; complex, small datasets ⇒ perfect for kernel methods!
  - Life sciences, medicine, ...

Recap: kernels

RKHS	$\mathcal{H} \ni f : \mathcal{X} \to \mathbb{R}$
Kernel function	$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
Kernel trick	$k(x, z) = \langle \phi(x), \phi(z) \rangle$
Feature map	$\phi : \mathcal{X} \to \mathcal{H}$
Representer theorem	$f(x) = \sum_i \alpha_i k(x, x_i), \quad \alpha_i \in \mathbb{R}$

Recap: kernels

RKHS	$\mathcal{H} \ni f : \mathcal{X} \to \mathbb{R}$
Kernel function	$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
Kernel trick	$k(x, z) = \langle \phi(x), \phi(z) \rangle$
Feature map	$\phi : \mathcal{X} \to \mathcal{H}$
Representer theorem	$f(x) = \sum_i \alpha_i k(x, x_i), \quad \alpha_i \in \mathbb{R}$

$\mathcal{X}$ doesn't have to be $\mathbb{R}^d$!

Overview

Kernels on text

Learning from natural language documents.

Basic concepts:

- **Word**: any sequence of basic alphabet surrounded by punctuation or spaces.
- **Document**: sequence of words
- **Corpus**: full set of documents
- **Dictionary**: set of terms occurring in corpus

Bag-of-words representation

BoW feature:

$$\phi_{BoW}(x) = [tf(t_1, x), tf(t_2, x), ..., tf(t_N, x)]^\top$$

In which

- x is a document
- t_j is a term in the dictionary
- $tf(t_j, x)$ gives the **term frequency** of t_j in document x

BoW kernel:

$$k_{BoW}(x, z) = \langle \phi_{BoW}(x), \phi_{BoW}(z) \rangle$$

Bag-of-words representation

Not very efficient, however there are things to do to improve:

- Tokenization: avoid computing over sparse high-dimensional feature vectors
- Stop word removal: exclude frequent but non-informative words
- Stemming: remove inflections and word form variation
- Normalization: reducing the effect of the length of documents; $\phi(x) = \frac{\phi(x)}{\|\phi(x)\|}$

Doesn't capture all useful information:

- order of words "you are" vs "are you"
- grammatical information is lost (verb, noun..)

Semantic kernels

Add semantic content by considering transformations

$$\phi_S(x) = \mathbf{S}\,\phi_{BoW}(x),$$

where the new features are now linear combinations of the old ones. (e.g. group related words, documents have non-zero similarity if they use terms from the same group.)

$$k_S(x, z) = \phi_{BoW}(x)^\top \mathbf{S}^\top \mathbf{S} \phi_{BoW}(z)$$

Semantic kernels

$$k_S(x, z) = \phi_{BoW}(x)^\top \mathbf{S}^\top \mathbf{S} \phi_{BoW}(z)$$

Semantic kernel can be decomposed further: write $\mathbf{S} = \mathbf{RP}$;

- $\mathbf{R}$ is diagonal matrix containing term weights or relevances
  - Rare words might carry more meaning; inverse weighting
- $\mathbf{P}$ is proximity matrix defining semantic spread between the terms
  - $\mathbf{P}_{ij} > 0$ when term i is related to term j; e.g. "tree", "spruce" and "pine"
  - Load this from a database (e.g. WordNet), or analyse co-occurrences from large documents.

String kernels[2]

Kernel on words/strings instead of collections of words.

Based on counting common subsequences; underlying feature map contains a feature for each possible substring.

x AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
x' TACCTAATTATGAAATTAAATTTCAGTGTGCTGATGGAAACGGAGAAGTC

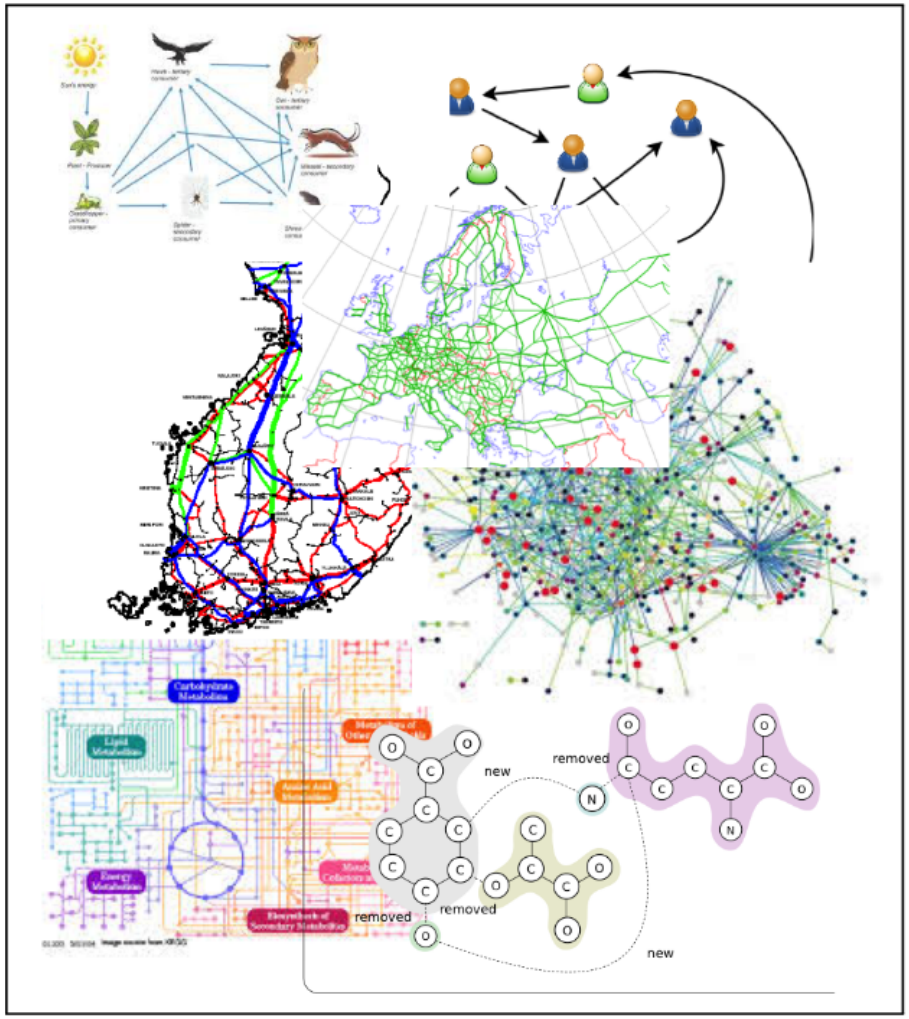
Multiple choices to tune the kernel:

- when do two subsequences match
- what kind of subsequences to consider
- how to weight matches

[2] Shawe-Taylor & Cristianini, chapter 11

Graphs are everywhere

Social networks, maps, communication channels, protein interaction networks, molecules, ...



Graphs are everywhere

Machine learning tasks on graphs:

- Given a graph, predict labels for its nodes
- Link prediction: given a set of nodes, predict which ones should be connected
- **Graph classification**
  - Drug discovery: given a candidate drug molecule (graph), predict if it will be active against a given type of cancer cell
  - Protein function prediction: given a 3D protein structure, predict its functional role

Graph kernels

A broad idea: try to find common elements (labels, paths, subgraphs, ...) and count their occurrences

Subgraphs & neighbourhoods:

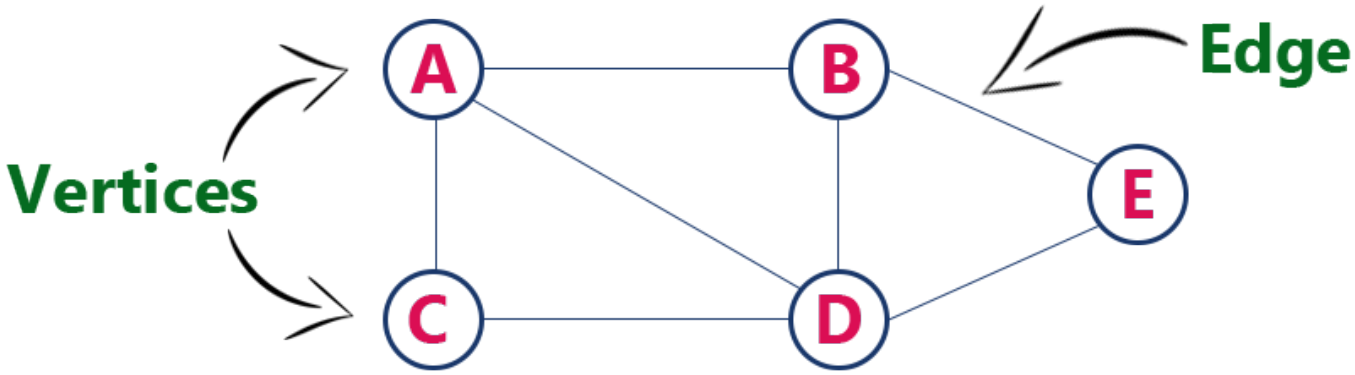
- Weisfeiler-Lehman kernel

Labels:

- Optimal assignment kernel

Paths / walks:

- Random walk kernel


Vertices
A
B
C
D
E
Edge

Weisfeiler-Lehman kernel

"How often the nodes' neighbourhoods match, up to depth d?"

$$k(G, G') = \sum_{i=0}^{d} \lambda_d \sum_{v \in V} \sum_{v' \in V'} \delta(relabel(v, i), relabel(v', i))$$

In which:

- G, G' are the graphs to compare
- d is depth parameter
- λ_d is parameter controlling importance of the level
- $\delta(\cdot, \cdot)$ returns 1 if its arguments are the same, 0 otherwise.
- $relabel(\cdot, i)$ returns a **new label for the node based on it's neighbour's labels on level $i-1$: its own label & a sorted list of neighbouring node's labels**
  - $i = 0$ just use original labels

Weisfeiler-Lehman kernel

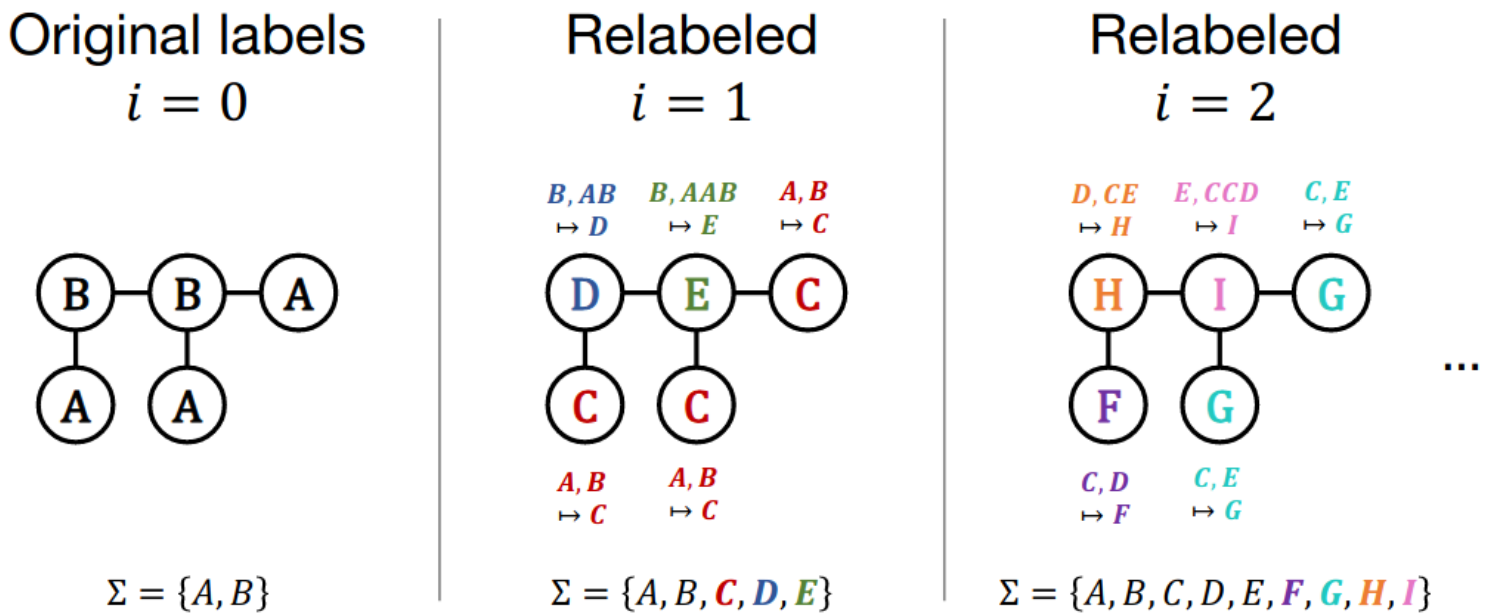


Image: Kriege, Johansson and Morris: *A Survey on Graph Kernels*, 2020

Optimal assignment(OA) kernel[4]

Find the best mapping between the nodes of two graphs.

$$k(G, G') = \max_{\pi \in \Pi_n} \sum_{i=1}^{n} \kappa(x_i, y_{\pi(i)})$$

In which:

- G, G' are the graphs to compare which have label sets $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_n\}$[3]
- π is a permutation from set of all possible permutations of n elements, Π_n
- κ is a kernel on labels

[3] If one graph has less labels than the other, fill the it up with dummies and define similarity to a dummy be always 0.

[4] Fröchlich et al, ICML 2005

optimal assignment(OA) kernel

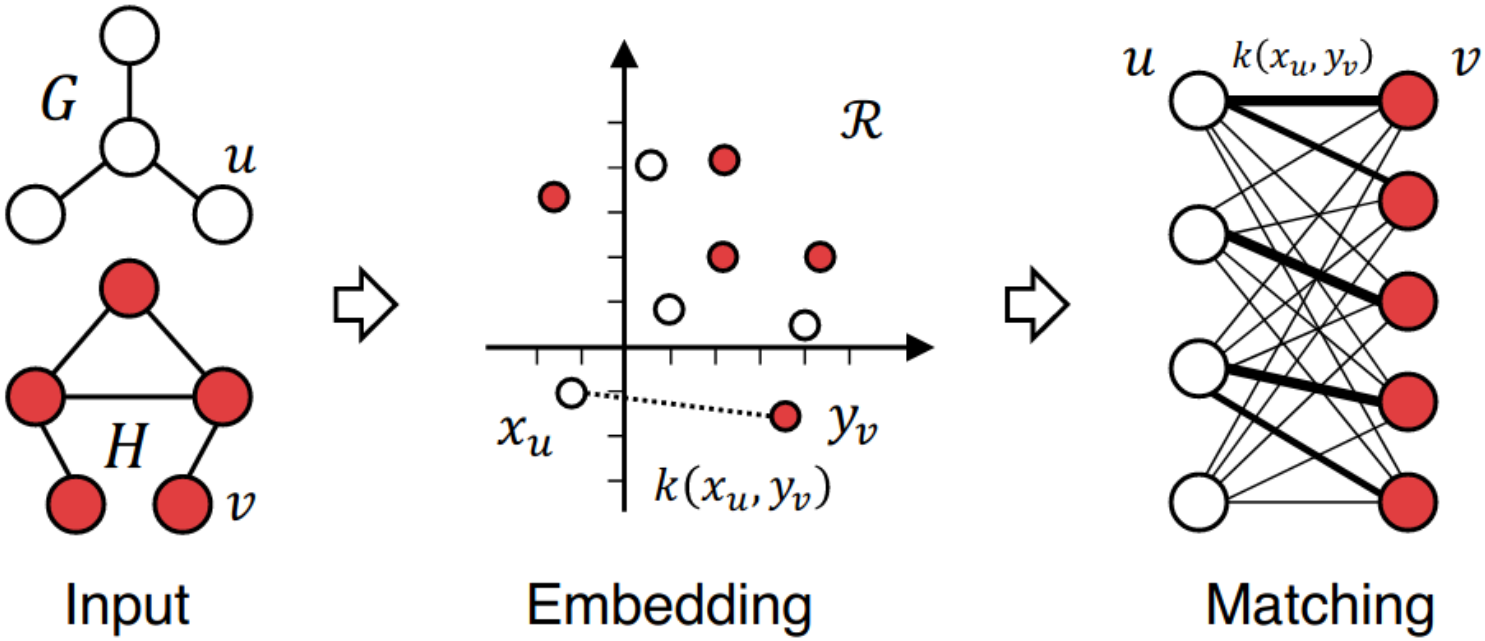


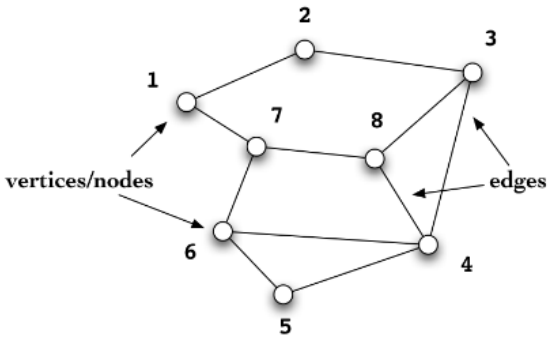
Image: Kriege, Johansson and Morris: *A Survey on Graph Kernels*, 2020

Random walk kernels[5]

Count common walks (sequences of adjacent nodes) in two graphs

- In an unlabeled graph two walks match if they have the same length
- In a labeled graph also the node and edge labels need to match

Number of walks of length k can be computed with adjacency matrix A raised to the kth power



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

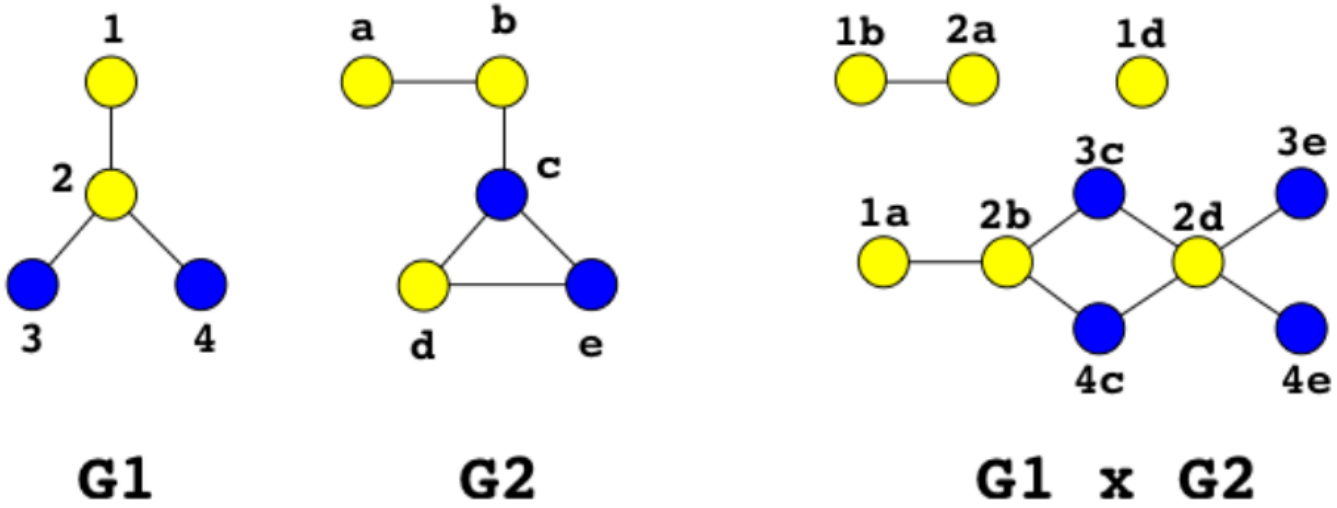
$$A^2 = \begin{bmatrix} 2 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 3 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 4 & 1 & 1 & 2 & 1 \\ 0 & 0 & 1 & 1 & 2 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 3 & 0 & 2 \\ 0 & 1 & 1 & 2 & 1 & 0 & 3 & 0 \\ 1 & 1 & 1 & 1 & 1 & 2 & 0 & 3 \end{bmatrix}$$

[5] Kashima et al., ICML 2003, Gärtner et al., COLT 2003

Random walk kernels

$G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2) \Rightarrow$ product graph $G_\times = (V_\times, E_\times)$;

- $V_\times = \{(u, v) : u \in V_1, v \in V_2, label(u) = label(v)\}$
  Make a new node for all pairs for which the labels match.
- $E_\times = \{((u_1, v_1), (u_2, v_2)) \in E_1 \times E_2 : (u_1, u_2) \in V_\times,$
  $(v_1, v_2) \in V_\times, label((u_1, u_2)) = label((v_1, v_2))\}$
  Two vertices in the direct product graph are adjacent iff the associated pairs of vertices are adjacent in original graphs.



*labels=colors

Random walk kernels

Use the adjacency matrix, $A_{\times}^{k}$, to simultaneously trace common **labeled** walks in the two original graphs with $G_{\times}$!

- Tracing a walk in the product graph corresponds to simultaneously tracing common walks in the two original graphs
- Ignore the labels in product graph; yet count only walks with matching labels.

Random walk kernels

Use the adjacency matrix, $A_\times^k$, to simultaneously trace common **labeled** walks in the two original graphs with $G_\times$!

- Tracing a walk in the product graph corresponds to simultaneously tracing common walks in the two original graphs
- Ignore the labels in product graph; yet count only walks with matching labels.

Random walk graph kernel: count all pairs of matching walks

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n A_\times^n \right]_{ij}$$

- Considers walks of *any* length! $[A_\times^n]_{ij}$ contains the number of walks of length n between the nodes i and j
- $0 \leq \lambda \leq 1$ is a decaying factor for the sum to converge
- Corresponds to an infinite-dimensional feature space

Random walk kernels

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n A_\times^n \right]_{ij} = \sum_{i,j=1}^{|V_\times|} s_{ij}$$

Random walk kernels

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n A_\times^n \right]_{ij} = \sum_{i,j=1}^{|V_\times|} s_{ij}$$

Geometric matrix series:

$$S = [s_{ij}]_{i,j=1}^{|V_\times|} = \sum_{n=0}^{\infty} \lambda^n A_\times^n = I_{|V_\times|} + \lambda A_\times + \lambda^2 A_\times^2 + \ldots$$

Random walk kernels

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n A_\times^n \right]_{ij} = \sum_{i,j=1}^{|V_\times|} s_{ij}$$

Geometric matrix series:

$$S = [s_{ij}]_{i,j=1}^{|V_\times|} = \sum_{n=0}^{\infty} \lambda^n A_\times^n = I_{|V_\times|} + \lambda A_\times + \lambda^2 A_\times^2 + \ldots$$

Multiply both sides with $\lambda A_\times$:

$$\lambda A_\times S = \lambda A_\times \textstyle\sum_{\mathbf{n=0}}^{\infty} \lambda^n A_\times^n$$

Random walk kernels

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n A_\times^n \right]_{ij} = \sum_{i,j=1}^{|V_\times|} s_{ij}$$

Geometric matrix series:

$$S = [s_{ij}]_{i,j=1}^{|V_\times|} = \sum_{n=0}^{\infty} \lambda^n A_\times^n = I_{|V_\times|} + \lambda A_\times + \lambda^2 A_\times^2 + \ldots$$

Multiply both sides with $\lambda A_\times$:

$$\lambda A_\times S = \lambda A_\times \sum_{\mathbf{n=0}}^{\infty} \lambda^n A_\times^n = \sum_{\mathbf{n=1}}^{\infty} \lambda^n A_\times^n = S - I_{|V_\times|}$$

Random walk kernels

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n A_\times^n \right]_{ij} = \sum_{i,j=1}^{|V_\times|} s_{ij}$$

Geometric matrix series:

$$S = [s_{ij}]_{i,j=1}^{|V_\times|} = \sum_{n=0}^{\infty} \lambda^n A_\times^n = I_{|V_\times|} + \lambda A_\times + \lambda^2 A_\times^2 + \ldots$$

Multiply both sides with $\lambda A_\times$:

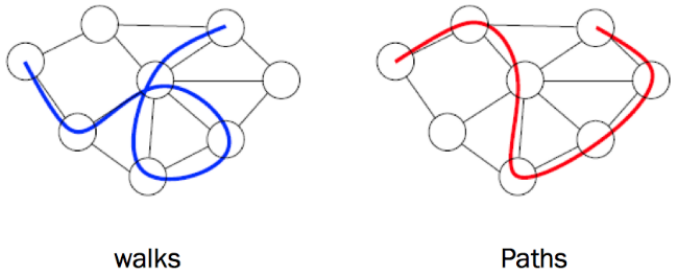
$$\lambda A_\times S = \lambda A_\times \sum_{\mathbf{n=0}}^{\infty} \lambda^n A_\times^n = \sum_{\mathbf{n=1}}^{\infty} \lambda^n A_\times^n = S - I_{|V_\times|}$$

$$\Rightarrow S = \left(I_{|V_\times|} - \lambda A_\times\right)^{-1}$$

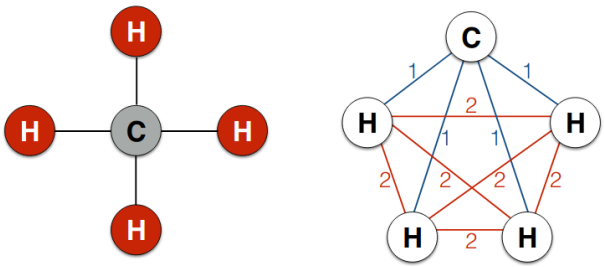
$$\Rightarrow k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\left(I_{|V_\times|} - \lambda A_\times\right)^{-1} \right]_{ij}$$

Shortest path kernels

RW kernels are not very efficient ($\mathcal{O}(n^6)$) and suffer from tottering $\Rightarrow$ from walks to paths



Shortest path kernels[6] much more efficient; $\mathcal{O}(n^4)$. (Uses Floyd-Warshall to find the paths).

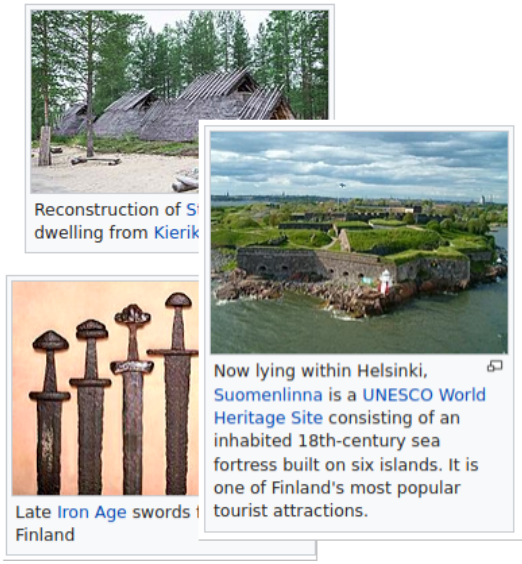


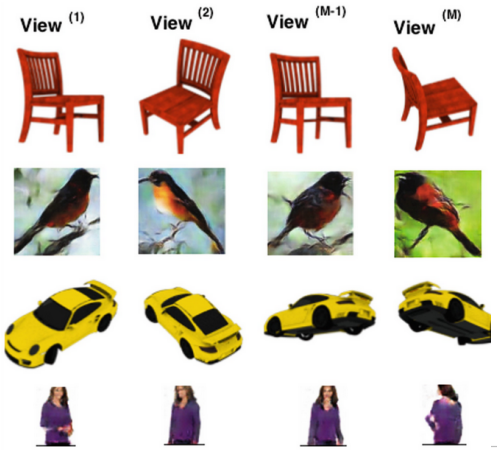
Transform input graph to shortest-path graph.
Kernel is obtained by comparing node and edge labels in the new graph.

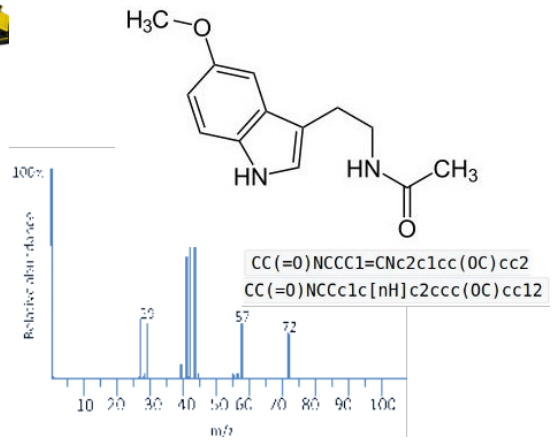
[6] Borgwardt and Kriegel, 2005

Overview

Multi-view data







Moi **bonjour**
hello `Guten Tag`

Integrating data sources

Early fusion:

- Combine the data sources first (human expert, simple vector concatenation..)
- Learn a single model from combined data

Intermediate fusion:

- Input all data sources separately
- Learn how to combine the sources during learning the model
  - **Multiple Kernel Learning**

Late fusion:

- Learn a model for each data source independently
- Combine the results after learning e.g. majority voting

Multiple kernel learning

Simple input data concatenation in feature spaces:

$$k(x, z) = \sum_{v=1}^{V} k^v(x, z)$$

Multiple Kernel Learning (MKL):

$$k(x, z) = \sum_{v=1}^{V} \alpha_v k^v(x, z),$$

in which the weights $\alpha_v > 0$ are learnt.

Multiple kernel learning

How to learn the combination weights?

- Two-step approach with kernel alignment, where weights are learned before the predictive model
- One-step approaches: learn the weights jointly with the learning problem
  - Note: usually iterates over learning the kernel weights and learning the decision function

MKL with kernel alignment[7]

Kernel alignment:

$$A(\mathbf{K}, \mathbf{G}) = \frac{\langle \mathbf{K}_c, \mathbf{G}_c \rangle_F}{\|\mathbf{K}_c\|_F \|\mathbf{G}_c\|_F}$$

in which $\mathbf{K}_c = \left(\mathbf{I} - \frac{1}{n}\mathbb{1}\mathbb{1}^\top\right)\mathbf{K}\left(\mathbf{I} - \frac{1}{n}\mathbb{1}\mathbb{1}^\top\right)$

Ideal (target) kernel for binary classification: $\mathbf{y}\mathbf{y}^\top$ with $y_i = -1, 1$.

A good kernel $\mathbf{K}$ for binary classification has large $A(\mathbf{K}, \mathbf{y}\mathbf{y}^\top)$.

[7] Cortes et al 2012

MKL with kernel alignment

ALIGN assigns the weights in MKL kernel to be the scores $A(\mathbf{K}^v, \mathbf{y}\mathbf{y}^\top)$:

$$k(x,z) = \sum_{v=1}^{V} \alpha_v k^v(x,z) = \sum_{v=1}^{V} A(\mathbf{K}^v, \mathbf{y}\mathbf{y}^\top) k^v(x,z),$$

ALIGNF problem:

$$\max_{\mathbf{d}} \frac{\left\langle \sum_{v=1}^{V} d_v \mathbf{K}^v, \mathbf{y}\mathbf{y}^\top \right\rangle_F}{\left\| \sum_{v=1}^{V} d_v \mathbf{K}^v \right\|_F \left\| \mathbf{y}\mathbf{y}^\top \right\|_F}, \quad s.t \left\| \mathbf{d} \right\|_2 = 1, d_i \geq 0$$

solved as a quadratic programming problem.

SimpleMKL[8]

- One-step approach: learn the model (such as SVM classifier) simultaneously with the kernel weights.
- Learns a linear combination

$$k_d(x_i, x_j) = \sum_{m=1}^{P} d_m k_m(x_i, x_j)$$

where kernel weights are constrained to convex combination

$$\sum_m d_m = 1, d_m \geq 0$$

- Bi-level optimization scheme with SVM solver as a wrapper

[8] Rakotomamonjy et al 2008

SimpleMKL: the idea

- Can be interpreted as learning a mixture of classifiers
- One base classifier for each kernel, with kernel weight d_m, example weights α_i:

$$f_m(\mathbf{x}) = \langle \mathbf{w}_m, \phi_m(\mathbf{x}) \rangle = \sum_i \alpha_i y_i d_m \kappa_m(\mathbf{x}, \mathbf{x}_i)$$

- Above $\mathbf{w}_m = \sum_i \alpha_i y_i d_m \phi_m(\mathbf{x}_i)$ obtained from Lagrangian duality
- The full model will use a mixture of base classifiers:

$$f(\mathbf{x}) = \sum_{m=1}^{P} f_m(\mathbf{x}) = \sum_i \alpha_i y_i d_m \kappa_m(\mathbf{x}, \mathbf{x}_i)$$

SimpleMKL: primal problem

$$\min_{\mathbf{w},b,\xi,d} \frac{1}{2}\sum_m \frac{1}{d_m}\|\mathbf{w}_m\|^2 + C\sum_i \xi_i$$

$$s.t.\ y_i\left(\sum_m \langle \mathbf{w}_m, \phi_m(\mathbf{x}_i)\rangle\ (+b)\right) \geq 1-\xi_i, i=1\ldots,\ell$$

$$\xi_i \geq 0, i=1\ldots,\ell$$

$$\sum_m d_m = 1, d_m \geq 0, m=1,\ldots,P$$

- Objective minimizes a linear combination of norms of weight vectors corresponding to different kernels plus slack for examples
- Constraints declare that the mixture classifier should achieve a large margin
- It is a convex problem (Rakotomamonjy, 2008)

SimpleMKL: bi-level optimization, primal

- Reformulation as a bi-level optimization problem

$$\min_{\boldsymbol{d} \in H} \; J(\boldsymbol{d})$$

$$s.t.\; J(\boldsymbol{d}) = \begin{cases} \min_{\mathbf{w},b,\xi} & \frac{1}{2}\sum_m \frac{1}{d_m} \|\mathbf{w}_m\|^2 + C\sum_i \xi_i \\ s.t. & y_i \left(\sum_m \langle \mathbf{w}_m, \phi_m(\mathbf{x}_i)\rangle \; (+b)\right) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1\ldots,\ell \end{cases}$$

- In outer loop optimize kernel weights $\boldsymbol{d}$
- Inner loop corresponds to a primal soft-margin SVM using a combined kernel with current kernel weights

SimpleMKL: bi-level optimization, dual

- We can plug in the dual of the SVM problem in the inner loop

$$\min_{\boldsymbol{d} \in H} \; J(\boldsymbol{d})$$

$$s.t.\; J(\boldsymbol{d}) = \begin{cases} \max_{\alpha} & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j \sum_m d_m \kappa_m(\mathbf{x}_i, \mathbf{x}_j) \\ s.t. & 0 \leq \alpha_i \leq C \\ & (\sum_i \alpha_i y_i = 0) \end{cases}$$

- In outer loop optimize kernel weights $\boldsymbol{d}$
- Inner loop corresponds to a dual soft-margin SVM using a combined kernel with current kernel weights

Overview

Structured output prediction

It's easy to predict functions $f : \mathcal{X} \to \mathbb{R}$ with kernel methods; $f \in \mathcal{H}$.

By extension also vector-valued functions $f : \mathcal{X} \to \mathbb{R}^p$ are easy.

We have seen that it's perfectly fine for $\mathcal{X}$ to be structured data and not just subset of $\mathbb{R}^d$.

What if output space $\mathcal{Y}$ in $f : \mathcal{X} \to \mathcal{Y}$ is structured?

Structured output prediction

- Want to find f
- Know how to map $\mathcal{Y}$ to RKHS $\mathcal{F}_y$ with ϕ_y

→ Learn $h : \mathcal{X} \to \mathcal{F}_y$

→ In general g cannot be found explicitly: search the set of $\mathcal{Y}$ for element that would give the predicted value in $\mathcal{F}_y$;

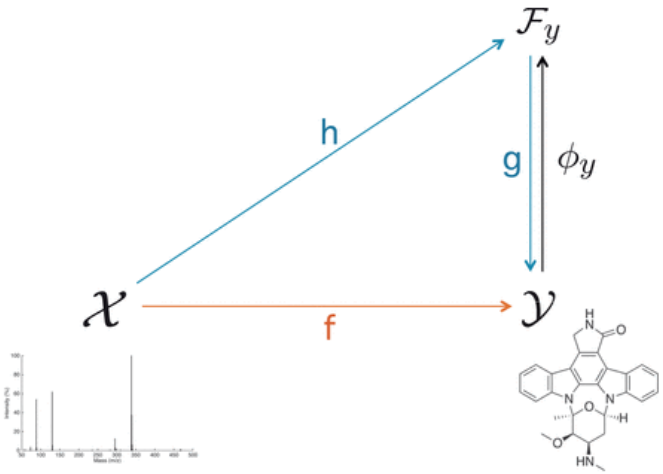


Image: Brouard et al: *Fast metabolite identification with Input Output Kernel Regression*, 2016

Conclusion

- Kernel methods are useful for many real-world learning problems that neural networks are not well applicable to.
- Kernels can handle structured inputs; kernels for text, graphs..
- Kernels are a natural choice for incorporating multiple views in a learning problem.
- Kernels can also be used in prediction problems when target of a learning problem is not vectorial, but structured.