Statistical learning with graphs

Jean-Philippe Vert
Jean-Philippe.Vert@ensmp.fr

Centre for Computational Biology
Ecole des Mines de Paris, ParisTech

Journées de Statistiques du Sud, Université de Nice Sophia Antipolis, April 11-14, 2007.

Outline

Outline

Outline

Part 1

Statistical Learning with Positive Definite Kernels

Outline

Overview

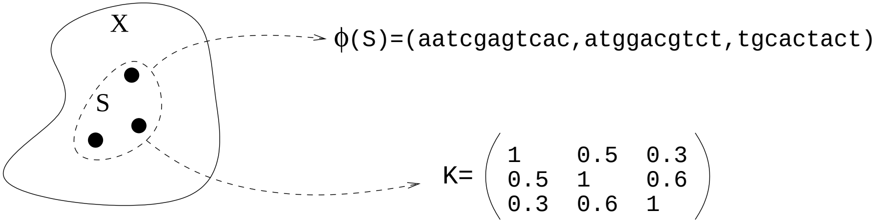
Motivations

- Develop versatile algorithms to process and learn from data
- No hypothesis made regarding the type of data (vectors, strings, graphs, images, ...)

The approach

- Develop methods based on pairwise comparisons.
- By imposing constraints on the pairwise comparison function (positive definite kernels), we obtain a nice general framework for learning from data.

Representation by pairwise comparisons



$$K = \begin{pmatrix} 1 & 0.5 & 0.3 \\ 0.5 & 1 & 0.6 \\ 0.3 & 0.6 & 1 \end{pmatrix}$$

Idea

- Define a “comparison function”: $K : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$.
- Represent a set of n data points $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ by the $n \times n$ matrix:

$$[K]_{ij} := K\left(\mathbf{x}_i, \mathbf{x}_j\right) .$$

Positive Definite (p.d.) Kernels

Definition

A positive definite (p.d.) kernel on the set $\mathcal{X}$ is a function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ symmetric:

$$\forall \left(\mathbf{x}, \mathbf{x}'\right) \in \mathcal{X}^2, \quad K\left(\mathbf{x}, \mathbf{x}'\right) = K\left(\mathbf{x}', \mathbf{x}\right),$$

and which satisfies, for all $N \in \mathbb{N}$, $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathcal{X}^N$ et $(a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K\left(\mathbf{x}_i, \mathbf{x}_j\right) \geq 0 .$$

General remarks

Remarks

- Equivalently, a kernel K is p.d. if and only if, for any $N \in \mathbb{N}$ and any set of points $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathcal{X}^N$, the similarity matrix $[K]_{ij} := K\left(\mathbf{x}_i, \mathbf{x}_j\right)$ is positive semidefinite.
- Complete modularity between the kernel (mapping a set of points to a matrix) and the algorithm (processing the matrix)
- Poor scalability w.r.t to the dataset size (n^2?)

Examples

Kernels for vectors

Classical kernels for vectors ($\mathcal{X} = \mathbb{R}^p$) include:

- The linear kernel

$$K_{lin}\left(\mathbf{x}, \mathbf{x}'\right) = \mathbf{x}^\top \mathbf{x}' \,.$$

- The polynomial kernel

$$K_{poly}\left(\mathbf{x}, \mathbf{x}'\right) = \left(\mathbf{x}^\top \mathbf{x}' + a\right)^d \,.$$

- The Gaussian RBF kernel:

$$K_{Gaussian}\left(\mathbf{x}, \mathbf{x}'\right) = \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}' \|^2}{2\sigma^2}\right) \,.$$

Examples

Kernels for vectors

Classical kernels for vectors ($\mathcal{X} = \mathbb{R}^p$) include:

- The linear kernel

$$K_{lin}\left(\mathbf{x}, \mathbf{x}'\right) = \mathbf{x}^\top \mathbf{x}' \,.$$

- The polynomial kernel

$$K_{poly}\left(\mathbf{x}, \mathbf{x}'\right) = \left(\mathbf{x}^\top \mathbf{x}' + a\right)^d \,.$$

- The Gaussian RBF kernel:

$$K_{Gaussian}\left(\mathbf{x}, \mathbf{x}'\right) = \exp\left(-\frac{\parallel \mathbf{x} - \mathbf{x}' \parallel^2}{2\sigma^2}\right) \,.$$

Examples

Kernels for vectors

Classical kernels for vectors ($\mathcal{X} = \mathbb{R}^p$) include:

- The linear kernel
$$K_{lin}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}' \,.$$
- The polynomial kernel
$$K_{poly}(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^\top \mathbf{x}' + a\right)^d \,.$$
- The Gaussian RBF kernel:
$$K_{Gaussian}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\parallel \mathbf{x} - \mathbf{x}' \parallel^2}{2\sigma^2}\right) \,.$$

P.d. kernels are inner products

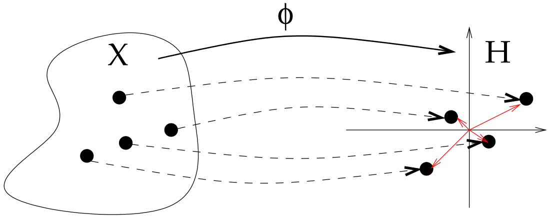
Theorem (Aronszajn, 1950)

K is a p.d. kernel on the set $\mathcal{X}$ if and only if there exists a Hilbert space $\mathcal{H}$ and a mapping

$$\Phi : \mathcal{X} \mapsto \mathcal{H} \, ,$$

such that, for any $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$:

$$K(\mathbf{x}, \mathbf{x}') = \left\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \right\rangle_{\mathcal{H}} \; .$$



Corollary: The kernel trick

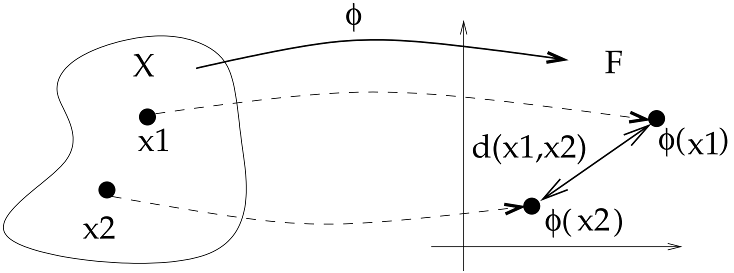
Kernel trick

Any algorithm to process finite-dimensional vectors that can be expressed only in terms of pairwise inner products can be applied to potentially infinite-dimensional vectors in the feature space of a p.d. kernel by replacing each inner product evaluation by a kernel evaluation.

Remarks

- The proof of this proposition is trivial, because the kernel is exactly the inner product in the feature space.
- This trick has huge practical applications, in particular to extend linear methods to non-linear settings and non-vector data.
- Vectors in the feature space are only manipulated implicitly, through pairwise inner products.

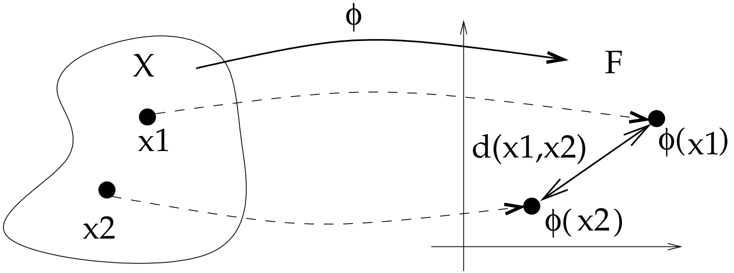
Kernel trick example: computing distances in the feature space



$$\begin{aligned} d_K(\mathbf{x}_1, \mathbf{x}_2)^2 &= \| \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \|_{\mathcal{H}}^2 \\ &= \langle \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} \\ &= \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_1) \rangle_{\mathcal{H}} + \langle \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} - 2 \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} \end{aligned}$$

$$d_K(\mathbf{x}_1, \mathbf{x}_2)^2 = K(\mathbf{x}_1, \mathbf{x}_1) + K(\mathbf{x}_2, \mathbf{x}_2) - 2K(\mathbf{x}_1, \mathbf{x}_2)$$

Kernel trick example: computing distances in the feature space



$$
\begin{aligned}
d_K(\mathbf{x}_1,\mathbf{x}_2)^2 &= \| \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \|_{\mathcal{H}}^2 \\
&= \langle \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} \\
&= \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_1) \rangle_{\mathcal{H}} + \langle \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} - 2 \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}}
\end{aligned}
$$

$$d_K(\mathbf{x}_1,\mathbf{x}_2)^2 = K(\mathbf{x}_1,\mathbf{x}_1) + K(\mathbf{x}_2,\mathbf{x}_2) - 2K(\mathbf{x}_1,\mathbf{x}_2)$$

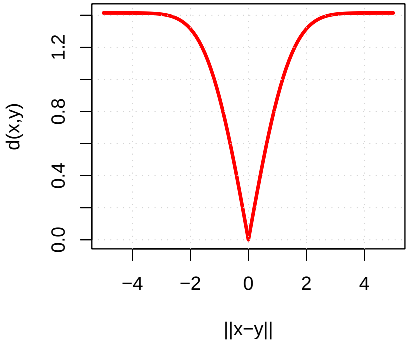
Distance for the Gaussian kernel

- The Gaussian kernel with bandwidth σ on $\mathbb{R}^d$ is:

$$K(\mathbf{x},\mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}},$$

- $K(\mathbf{x},\mathbf{x}) = 1 = \|\Phi(\mathbf{x})\|_{\mathcal{H}}^2$, so all points are on the unit sphere in the feature space.
- The distance between the images of two points **x** and **y** in the feature space is given by:

$$d_K(\mathbf{x},\mathbf{y}) = \sqrt{2\left[1 - e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}}\right]}$$



Reproducing kernel Hilbert space

Definition

Let $\mathcal{X}$ be a set and $\mathcal{H} \subset \mathbb{R}^{\mathcal{X}}$ be a class of functions forming a (real) Hilbert space with inner product $\langle ., . \rangle_{\mathcal{H}}$. The function $K : \mathcal{X}^2 \mapsto \mathbb{R}$ is called a reproducing kernel (r.k.) of $\mathcal{H}$ if

1. $\mathcal{H}$ contains all functions of the form

$$\forall \mathbf{x} \in \mathcal{X}, \quad K_{\mathbf{x}} : \mathbf{t} \mapsto K(\mathbf{x}, \mathbf{t}) \, .$$

2. For every $\mathbf{x} \in \mathcal{X}$ and $f \in \mathcal{H}$ the reproducing property holds:

$$f(\mathbf{x}) = \langle f, K_{\mathbf{x}} \rangle_{\mathcal{H}} \, .$$

If a r.k. exists, then $\mathcal{H}$ is called a reproducing kernel Hilbert space (RKHS).

Equivalence between positive definite and reproducing kernels

Theorem (Aronszajn, 1950)

K is a p.d. kernel if and only if there exists a RKHS having K as r.k.

Corollary

For any p.d. kernel K, let $\mathcal{H}$ be its RKHS. Define:

$$\begin{aligned} \Phi \ &: \mathcal{X} \to \mathcal{H} \,, \\ \mathbf{x} &\mapsto K_{\mathbf{x}} \,. \end{aligned}$$

We then get:

$$\begin{aligned} \left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right)\right\rangle_{\mathcal{H}} &= \langle K_{\mathbf{x}}, K_{\mathbf{x}'}\rangle_{\mathcal{H}} \\ &= K_{\mathbf{x}}\left(\mathbf{x}\right) \\ &= K\left(\mathbf{x},\mathbf{x}'\right) \ . \qquad \square \end{aligned}$$

Equivalence between positive definite and reproducing kernels

Theorem (Aronszajn, 1950)

K is a p.d. kernel if and only if there exists a RKHS having K as r.k.

Corollary

For any p.d. kernel K, let $\mathcal{H}$ be its RKHS. Define:

$$\begin{aligned}\Phi\ &: \mathcal{X} \to \mathcal{H}\,,\\ \mathbf{x} &\mapsto K_{\mathbf{x}}\,.\end{aligned}$$

We then get:

$$\begin{aligned}\left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right)\right\rangle_{\mathcal{H}} &= \langle K_{\mathbf{x}}, K_{\mathbf{x}'}\rangle_{\mathcal{H}}\\ &= K_{\mathbf{x}}\left(\mathbf{x}\right)\\ &= K\left(\mathbf{x}, \mathbf{x}'\right)\ . \qquad \square\end{aligned}$$

RKHS of a p.d. kernel

Explicit construction of the RKHS

- If K is p.d., then the RKHS $\mathcal{H}$ is the vector subspace of $\mathbb{R}^{\mathcal{X}}$ spanned by the functions $\{K_{\mathbf{x}}\}_{\mathbf{x}\in\mathcal{X}}$ (and their pointwise limits).
- For any $f, g \in \mathcal{H}_0$, given by:

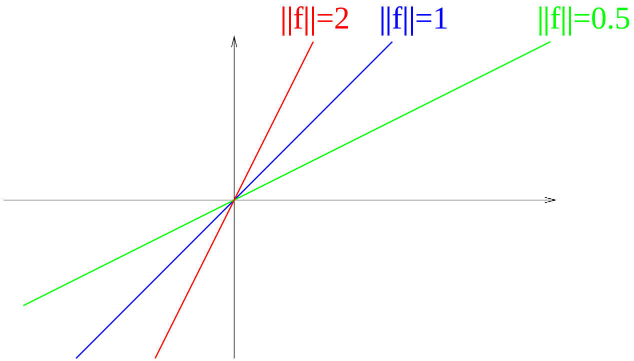
$$f = \sum_i a_i K_{\mathbf{x}_i} \,, \quad g = \sum_j b_j K_{\mathbf{y}_j} \,,$$

the inner product is given by:

$$\langle f, g \rangle_{\mathcal{H}_0} := \sum_{i,j} a_i b_j K\left(\mathbf{x}_i, \mathbf{y}_j\right) \,, \quad \| f \|^2_{\mathcal{H}_0} = \sum_{i,j} a_i a_j K\left(\mathbf{x}_i, \mathbf{x}_j\right) \,.$$

Example : RKHS of the linear kernel

$$\begin{cases} K(\mathbf{x}, \mathbf{x}') & = \mathbf{x}^\top \mathbf{x}' \, . \\ f(\mathbf{x}) & = w^\top \mathbf{x} \, , \\ \| f \|_{\mathcal{H}} & = \| w \|_2 \, . \end{cases}$$



Examples: RKHS of the Gaussian RBF kernel

$$
\begin{aligned}
K_{Gaussian}\left(\mathbf{x}, \mathbf{x}'\right) &= \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}' \|^2}{2\sigma^2}\right) \,, \\
f\left(\mathbf{x}\right) &= \sum_{i=1}^{n} \alpha_i \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}_i \|^2}{2\sigma^2}\right) \,, \\
\| f \|_{\mathcal{H}}^2 &= \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j \exp\left(-\frac{\| \mathbf{x}_i - \mathbf{x}_j \|^2}{2\sigma^2}\right) \\
&= \int \left| \hat{f}(\omega) \right|^2 e^{\frac{\sigma^2\omega^2}{2}} d\omega \,.
\end{aligned}
$$

Smoothness functional

A simple inequality

- By Cauchy-Schwarz we have, for any function $f \in \mathcal{H}$ and any two points $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$:

$$
\begin{aligned}
\left| f(\mathbf{x}) - f(\mathbf{x}') \right| &= \left| \langle f, K_{\mathbf{x}} - K_{\mathbf{x}'} \rangle_{\mathcal{H}} \right| \\
&\leq \| f \|_{\mathcal{H}} \times \| K_{\mathbf{x}} - K_{\mathbf{x}'} \|_{\mathcal{H}} \\
&= \| f \|_{\mathcal{H}} \times d_K(\mathbf{x}, \mathbf{x}') \ .
\end{aligned}
$$

- The norm of a function in the RKHS controls how fast the function varies over $\mathcal{X}$ with respect to the geometry defined by the kernel (Lipschitz with constant $\| f \|_{\mathcal{H}}$).

Important message

$$\text{Small norm} \implies \text{slow variations.}$$

A useful property

Representer theorem (Kimeldorf and Wahba, 1971)

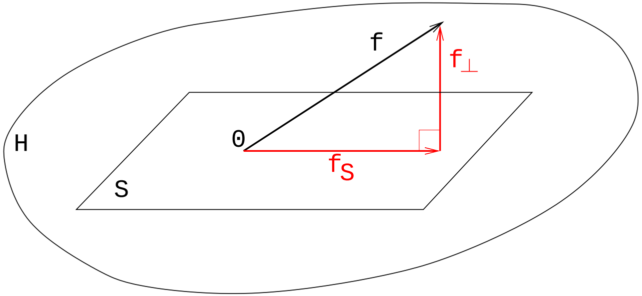
- Let $\mathcal{X}$ be a set endowed with a p.d. kernel K, $\mathcal{H}_K$ the corresponding RKHS, and $\mathcal{S} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\} \subset \mathcal{X}$ a finite set of points in $\mathcal{X}$.
- Let $\Psi : \mathbb{R}^{n+1} \to \mathbb{R}$ be a function of $n+1$ variables, strictly increasing with respect to the last variable.
- Then, any solution to the optimization problem:

$$\min_{f \in \mathcal{H}_K} \Psi\left(f(\mathbf{x}_1), \cdots, f(\mathbf{x}_n), \| f \|_{\mathcal{H}_K}\right),$$

admits a representation of the form:

$$\forall \mathbf{x} \in \mathcal{X}, \quad f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}).$$

Representer theorem: proof



- $\mathcal{S} = \text{span}\left\{K_{\mathbf{x}_1}, \ldots, K_{\mathbf{x}_n}\right\}$
- $f_\perp(\mathbf{x}_i) = \langle f_\perp, K_{\mathbf{x}_i}\rangle_{\mathcal{H}_K} = 0 \implies f(\mathbf{x}_i) = f_{\mathcal{S}}(\mathbf{x}_i)$ for $i = 1, \ldots, n$.
- $\| f \|_{\mathcal{H}_K} > \| f_{\mathcal{S}} \|_{\mathcal{H}_K}$ if $f_\perp \neq 0$. (Pythagoras) $\square$

Remarks

Practical and theoretical consequences

Often the function Ψ has the form:

$$\Psi\left(f(\mathbf{x}_1), \cdots, f(\mathbf{x}_n), \| f \|_{\mathcal{H}_K}\right) = c\left(f(\mathbf{x}_1), \cdots, f(\mathbf{x}_n)\right) + \lambda\Omega\left(\| f \|_{\mathcal{H}_K}\right)$$

where $c(.)$ measures the “fit” of f to a given problem (regression, classification, dimension reduction, ...) and Ω is strictly increasing.
This formulation has two important consequences:

- Theoretically, the minimization will enforce the norm $\| f \|_{\mathcal{H}_K}$ to be “small”, which can be beneficial by ensuring a sufficient level of smoothness for the solution (regularization effect).
- Practically, we know by the representer theorem that the solution lives in a subspace of dimension n, which can lead to efficient algorithms although the RKHS itself can be of infinite dimension.

Outline

Learning from data

General setting

- Observation: $\{z_1, \ldots, z_n\}$ where $z_i = (\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
- Goal: learn a function $f : \mathcal{X} \to \mathbb{R}$
- Examples: density estimation, pattern recognition, regression, outlier detection, clustering, compression, low-dimensional embedding...

Learning from data

Empirical risk minimization (ERM)

1. Define a loss function $l(f,z)$ and a space of functions $\mathcal{F}$.
2. Minimize the empirical average loss over $\mathcal{F}$:

$$\hat{f} \in \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} l(f, z_i) \,.$$

General properties of ERM

- If $\mathcal{F}$ is not "too large" then the ERM is consistent ($\hat{f}$ is close to the best possible $f \in \mathcal{F}$ as the number of observations increases).
- If $\mathcal{F}$ is not "too small" then the best possible $f \in \mathcal{F}$ is a "good" solution.
- Challenge: choose a "small" $\mathcal{F}$ that contains "good" functions.

Learning from data

Empirical risk minimization (ERM)

1. Define a loss function $l(f,z)$ and a space of functions $\mathcal{F}$.
2. Minimize the empirical average loss over $\mathcal{F}$:

$$\hat{f} \in \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} l(f, z_i) \,.$$

General properties of ERM

- If $\mathcal{F}$ is not "too large" then the ERM is consistent ($\hat{f}$ is close to the best possible $f \in \mathcal{F}$ as the number of observations increases).
- If $\mathcal{F}$ is not "too small" then the best possible $f \in \mathcal{F}$ is a "good" solution.
- Challenge: choose a "small" $\mathcal{F}$ that contains "good" functions.

Learning with kernels

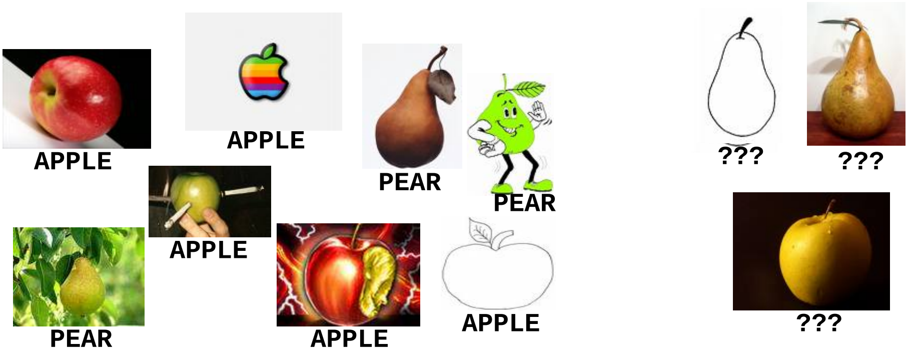
ERM in RKHS

- Take $\mathcal{F}$ to be a ball in the RKHS:

$$\mathcal{F}_B = \{ f \in \mathcal{H} \, : \, \| f \|_{\mathcal{H}} \leq B \} \, .$$

- Advantage: by controlling the "size" of $\mathcal{F}$ (related to B) the ERM principle works (consistency and theoretical rates of convergence).
- The kernel should be chosen s.t. some "good" functions have a small RKHS norm.

Example: pattern recognition



- Input variables $\mathbf{x} \in \mathcal{X}$
- Output $y \in \{-1, 1\}$.
- Training set $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$.

Large-margin classifiers

General setting

- For pattern recognition $\mathcal{Y} = \{-1, 1\}$.
- Goal: estimate a function $f : \mathcal{X} \to \mathbb{R}$ to predict **y** from the sign of $f(\mathbf{x})$
- The margin for a pair $(\mathbf{x}, \mathbf{y})$ is $\mathbf{y}f(\mathbf{x})$.
- Focusing on large margins ensures that $f(\mathbf{x})$ has the same sign as **y** and a large absolute value (confidence).
- Leads to a loss function

$$l(f, (\mathbf{x}, \mathbf{y})) = \phi(\mathbf{y}f(\mathbf{x})) \,,$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is non-increasing.

ERM in for large-margin classifiers: Theory

Theoretical results

- The ERM estimator $\hat{f}_n$ solves:

$$\begin{cases} \min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \phi\left(\mathbf{y}_i f\left(\mathbf{x}_i\right)\right) \\ \text{subject to } \| f \|_{\mathcal{H}} \leq B\,. \end{cases}$$

- Let P an unknown distribution over $\mathcal{X} \times \mathcal{Y}$, assume $\mathcal{S} = (\mathbf{x}_i, y_i)_{i=1,\ldots,n}$ i.i.d. according to P.
- Assume K upper bounded by κ and ϕ Lipschitz with constant L_ϕ.
- For the ϕ-risk $R_\phi(f) = \mathbf{E}\phi\left(Yf\left(X\right)\right)$ we have:

$$\mathbf{E}R_\phi\left(\hat{f}_n\right) \leq \inf_{f \in \mathcal{F}_B} R_\phi(f) + \frac{8 L_\phi \kappa B}{\sqrt{n}}\,.$$

ERM in for large-margin classifiers: Practice

Reformulation as penalized minimization

- We must solve the constrained minimization problem:

$$\begin{cases} \min_{f\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n}\phi\left(\mathbf{y}_i f\left(\mathbf{x}_i\right)\right) \\ \text{subject to } \| f \|_{\mathcal{H}} \leq B\,. \end{cases}$$

- To make this practical we assume that ϕ is convex.
- The problem is then a convex problem in f for which strong duality holds. In particular f solves the problem if and only if it solves for some dual parameter λ the unconstrained problem:

$$\min_{f\in\mathcal{H}}\left\{\frac{1}{n}\sum_{i=1}^{n}\phi\left(\mathbf{y}_i f\left(\mathbf{x}_i\right)\right)+\lambda\| f \|_{\mathcal{H}}^2\right\},$$

and complimentary slackness holds ($\lambda = 0$ or $\| f \|_{\mathcal{H}} = B$).

Optimization in RKHS

- By the representer theorem, the solution of the unconstrained problem can be expanded as:

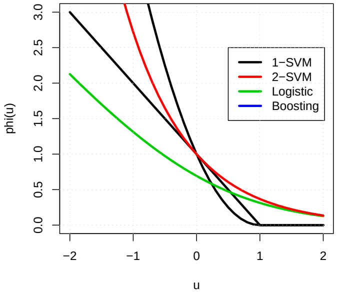
$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \; .$$

- Plugging into the original problem we obtain the following unconstrained and convex optimization problem in $\mathbb{R}^n$:

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \left\{ \frac{1}{n} \sum_{i=1}^{n} \phi\left(\mathbf{y}_i \sum_{j=1}^{n} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \right) + \lambda \sum_{i,j=1}^{n} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \right\} \; .$$

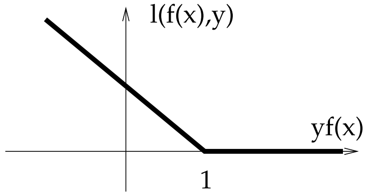
- This can be implemented using general packages for convex optimization or specific algorithms (e.g., for SVM).

Loss function examples



Method	$\phi(u)$
Kernel logistic regression	$\log\left(1+e^{-u}\right)$
Support vector machine (1-SVM)	$\max\left(1-u,0\right)$
Support vector machine (2-SVM)	$\max\left(1-u,0\right)^2$
Boosting	e^{-u}

Example: Support vector machines



- The loss function is the hinge loss:

$$\phi_{\text{hinge}}(u) = \max(1-u, 0) \; .$$

- SVM solve the problem:

$$\min_{f \in \mathcal{H}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \phi_{\text{hinge}}\left(\mathbf{y}_i f\left(\mathbf{x}_i\right)\right) + \lambda \| f \|_{\mathcal{H}}^2 \right\} \; .$$

SVM reformulation

The classifier is:

$$\forall \mathbf{x} \in \mathcal{X}, \quad f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}, \mathbf{x}_i) \ ,$$

where α is the solution of the following QP:

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^d} 2 \sum_{i=1}^{n} \alpha_i y_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \ ,$$

subject to:

$$0 \leq y_i \alpha_i \leq \frac{1}{n\lambda}, \quad \text{for } i = 1, \ldots, n \, .$$

Kernel methods: Summary

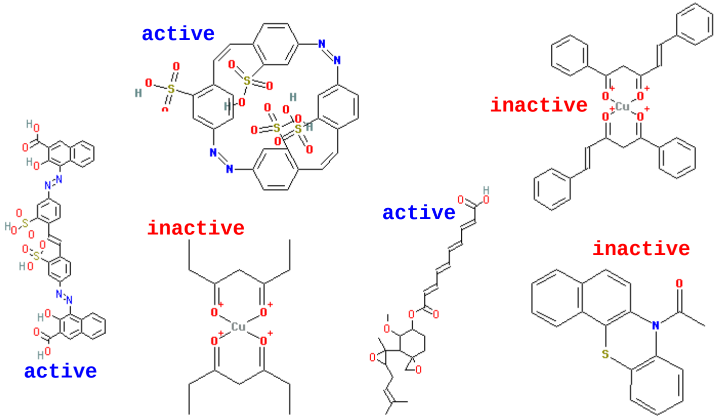
- 3 ways to map $\mathcal{X}$ to a Hilbert space:
  1. Explicitly define and compute $\Phi : \mathcal{X} \to \mathcal{H}$
  2. Define a p.d. kernel over $\mathcal{X}$
  3. Define a RKHS over $\mathcal{X}$
- The kernel trick allows to extend many linear algorithms to non-linear settings and to general data (even non-vectorial).
- The norm in the RKHS can be used as regularization for empirical risk minimization. This is theoretically justified and leads to efficient algorithms (often finite-dimensional convex problem thanks to the representer theorem).
- We are now ready to learn with graphs by defining positive definite kernels for graphs!

Part 2

Kernels for Graphs

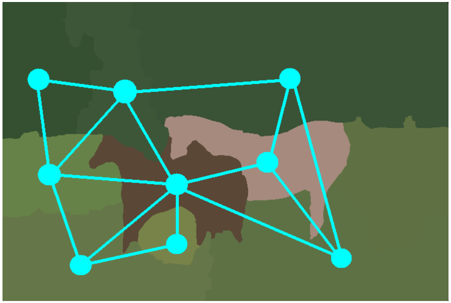
Outline

Chemoinformatics and QSAR



NCI AIDS screen results (from http://cactus.nci.nih.gov).

Image retrieval and classification



From Harchaoui and Bach (2007).

Graph kernel

Notations

- A directed graph is a pair $G = (V, E)$ with V finite (vertices) and $E \subset V \times V$ (edges).
- A graph is labeled if a label from a set of labels $\mathcal{A}$ is assigned to each vertex and/or edge.
- Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic (denoted $G_1 \simeq G_2$) if there exists a bijection between V_1 and V_2 that preserves edges and labels.

Definition

- We note $\mathcal{G}$ the quotient set of the set of all labelled graphs with respect to isomorphism.
- A graph kernel is a p.d. kernel over $\mathcal{G}$.

Graph kernel

Notations

- A directed graph is a pair $G = (V, E)$ with V finite (vertices) and $E \subset V \times V$ (edges).
- A graph is labeled if a label from a set of labels $\mathcal{A}$ is assigned to each vertex and/or edge.
- Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic (denoted $G_1 \simeq G_2$) if there exists a bijection between V_1 and V_2 that preserves edges and labels.

Definition

- We note $\mathcal{G}$ the quotient set of the set of all labelled graphs with respect to isomorphism.
- A graph kernel is a p.d. kernel over $\mathcal{G}$.

Outline

Expressiveness vs Complexity

Definition: Complete graph kernels

A graph kernel is complete if it separates non-isomorphic graphs, i.e.:

$$\forall G_1, G_2 \in \mathcal{G}, \quad d_K(G_1, G_2) = 0 \implies G_1 \simeq G_2 \, .$$

Equivalently, $\Phi(G_1) \neq \Phi(G_1)$ if G_1 and G_2 are not isomorphic.

Expressiveness vs Complexity trade-off

- If a graph kernel is not complete, then there is no hope to learn all possible functions over $\mathcal{G}$: the kernel is not expressive enough.
- On the other hand, kernel computation must be tractable, i.e., no more than polynomial (with small degree) for practical applications.
- Can we define tractable and expressive graph kernels?

Expressiveness vs Complexity

Definition: Complete graph kernels

A graph kernel is complete if it separates non-isomorphic graphs, i.e.:

$$\forall G_1, G_2 \in \mathcal{G}, \quad d_K(G_1, G_2) = 0 \implies G_1 \simeq G_2 \,.$$

Equivalently, $\Phi(G_1) \neq \Phi(G_1)$ if G_1 and G_2 are not isomorphic.

Expressiveness vs Complexity trade-off

- If a graph kernel is not complete, then there is no hope to learn all possible functions over $\mathcal{G}$: the kernel is not expressive enough.
- On the other hand, kernel computation must be tractable, i.e., no more than polynomial (with small degree) for practical applications.
- Can we define tractable and expressive graph kernels?

Complexity of graph algorithms

Known facts

- *Are G_1 and G_2 isomorphic?*
  The graph isomorphism problem is in NP. It is believed to lie between P and NP-complete. No known polynomial-time algorithm exists.
- *Is G_1 isomorphic to a subgraph of G_2?*
  The subgraph isomorphism problem is NP-complete.
- *Does G contain a sequence of adjacent vertices and edges that contains every vertex and edge exactly once?*
  The Hamiltonian path problem is NP-complete.

Complexity of graph algorithms

Known facts

- *Are G_1 and G_2 isomorphic?*
  The graph isomorphism problem is in NP. It is believed to lie between P and NP-complete. No known polynomial-time algorithm exists.
- *Is G_1 isomorphic to a subgraph of G_2?*
  The subgraph isomorphism problem is NP-complete.
- *Does G contain a sequence of adjacent vertices and edges that contains every vertex and edge exactly once?*
  The Hamiltonian path problem is NP-complete.

Complexity of graph algorithms

Known facts

- *Are G_1 and G_2 isomorphic?*
  The graph isomorphism problem is in NP. It is believed to lie between P and NP-complete. No known polynomial-time algorithm exists.
- *Is G_1 isomorphic to a subgraph of G_2?*
  The subgraph isomorphism problem is NP-complete.
- *Does G contain a sequence of adjacent vertices and edges that contains every vertex and edge exactly once?*
  The Hamiltonian path problem is NP-complete.

Complexity of complete kernels

Proposition (Gärtner et al., 2003)

Computing any complete graph kernel is at least as hard as the graph isomorphism problem.

Proof

- For any kernel K the complexity of computing d_K is the same as the complexity of computing K, because:

$$d_K(G_1, G_2)^2 = K(G_1, G_1) + K(G_2, G_2) - 2K(G_1, G_2) .$$

- If K is a complete graph kernel, then computing d_K solves the graph isomorphism problem ($d_K(G_1, G_2) = 0$ iff $G_1 \simeq G_2$). $\square$

Complexity of complete kernels

Proposition (Gärtner et al., 2003)

Computing any complete graph kernel is at least as hard as the graph isomorphism problem.

Proof

- For any kernel K the complexity of computing d_K is the same as the complexity of computing K, because:

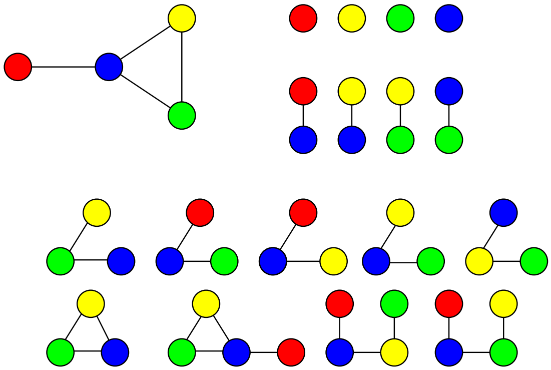
$$d_K(G_1, G_2)^2 = K(G_1, G_1) + K(G_2, G_2) - 2K(G_1, G_2)\,.$$

- If K is a complete graph kernel, then computing d_K solves the graph isomorphism problem ($d_K(G_1, G_2) = 0$ iff $G_1 \simeq G_2$). $\square$

Subgraphs

Definition

A subgraph of a graph (V, E) is a connected graph (V', E') with $V' \subset V$ and $E' \subset E$.



Subgraph kernel

Definition

- Let $(\lambda_G)_{G\in\mathcal{G}}$ a set or nonnegative real-valued weights
- For any graph $G \in \mathcal{G}$, let

$$\forall H \in \mathcal{G}, \quad \Phi_H(G) = \left| \left\{ G' \text{ is a subgraph of } G \,:\, G' \simeq H \right\} \right| .$$

- The subgraph kernel between any two graphs G_1 and $G_2 \in \mathcal{G}$ is defined by:

$$K_{subgraph}(G_1, G_2) = \sum_{H\in\mathcal{G}} \lambda_H \Phi_H(G_1)\Phi_H(G_2) .$$

Subgraph kernel complexity

Proposition (Gärtner et al., 2003)

Computing the subgraph kernel is NP-hard.

Proof (1/2)

- Let P_n be the path graph with n edges.
- The vectors $\Phi(P_1), \ldots, \Phi(P_n)$ are linearly independent, therefore:

$$e_{P_n} = \sum_{i=1}^{n} \alpha_i \Phi(P_i) \,.$$

- The coefficients α_i can be found in polynomial time (solving a $n \times n$ triangular system).

Subgraph kernel complexity

Proposition (Gärtner et al., 2003)

Computing the subgraph kernel is NP-hard.

Proof (1/2)

- Let P_n be the path graph with n edges.
- The vectors $\Phi(P_1), \ldots, \Phi(P_n)$ are linearly independent, therefore:

$$e_{P_n} = \sum_{i=1}^{n} \alpha_i \Phi(P_i) \,.$$

- The coefficients α_i can be found in polynomial time (solving a $n \times n$ triangular system).

Subgraph kernel complexity

Proposition (Gärtner et al., 2003)

Computing the subgraph kernel is NP-hard.

Proof (2/2)

- If G is a graph with n vertices, then it has a path that visits each node exactly once (Hamiltonian path) if and only if $\Phi(G)^\top e_n > 0$, i.e.,

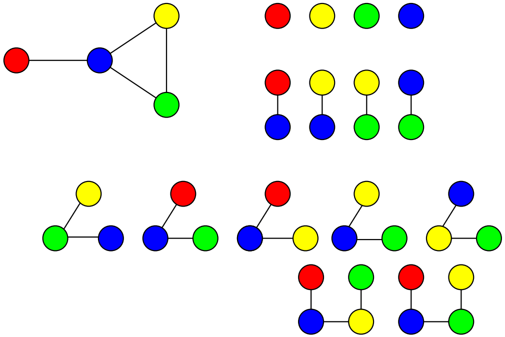
$$\sum_{i=1}^{n} \alpha_i K_{subgraph}(G, P_i) > 0 \,.$$

- The decision problem whether a graph has a Hamiltonian path is NP-complete. $\square$

Paths

Definition

- A path of a graph (V, E) is sequence of distinct vertices $v_1, \ldots, v_n \in V$ $(i \neq j \implies v_i \neq v_j)$ such that $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, n-1$.
- Equivalently the paths are the linear subgraphs.



Path kernel

Definition

The path kernel is the subgraph kernel restricted to paths, i.e.,

$$K_{path}(G_1, G_2) = \sum_{H \in \mathcal{P}} \lambda_H \Phi_H(G_1) \Phi_H(G_2) \,,$$

where $\mathcal{P} \subset \mathcal{G}$ is the set of path graphs.

Proposition (Gärtner et al., 2003)

Computing the path kernel is NP-hard.

Proof

Same as the subgraph kernel. $\square$

Path kernel

Definition

The path kernel is the subgraph kernel restricted to paths, i.e.,

$$K_{path}(G_1, G_2) = \sum_{H \in \mathcal{P}} \lambda_H \Phi_H(G_1) \Phi_H(G_2) \,,$$

where $\mathcal{P} \subset \mathcal{G}$ is the set of path graphs.

Proposition (Gärtner et al., 2003)

Computing the path kernel is NP-hard.

Proof

Same as the subgraph kernel. □

Path kernel

Definition

The path kernel is the subgraph kernel restricted to paths, i.e.,

$$K_{path}(G_1, G_2) = \sum_{H \in \mathcal{P}} \lambda_H \Phi_H(G_1) \Phi_H(G_2) \,,$$

where $\mathcal{P} \subset \mathcal{G}$ is the set of path graphs.

Proposition (Gärtner et al., 2003)

Computing the path kernel is NP-hard.

Proof

Same as the subgraph kernel. $\square$

Summary

Expressiveness vs Complexity trade-off

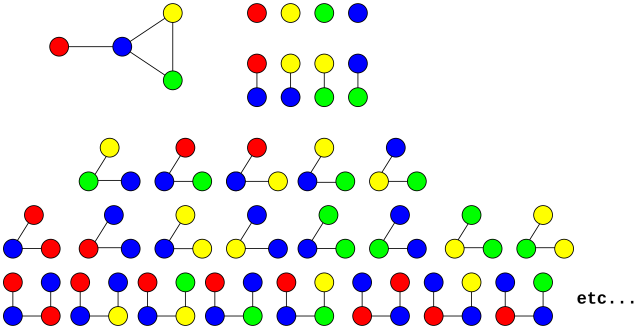
- It is intractable to compute complete graph kernels.
- It is intractable to compute the subgraph kernels.
- Restricting subgraphs to be linear does not help: it is also intractable to compute the path kernel.
- One approach to define polynomial time computable graph kernels is to have the feature space be made up of graphs homomorphic to subgraphs, e.g., to consider walks instead of paths.

Outline

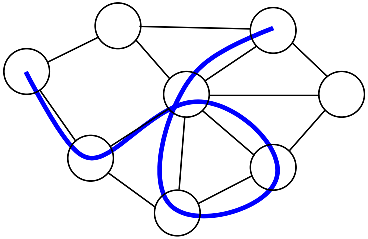
Walks

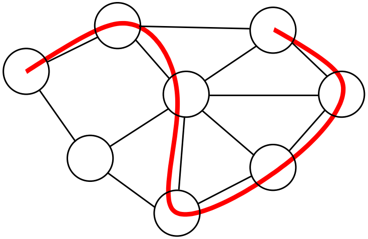
Definition

- A walk of a graph (V, E) is sequence of $v_1, \ldots, v_n \in V$ such that $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, n-1$.
- We note $\mathcal{W}_n(G)$ the set of walks with n vertices of the graph G, and $\mathcal{W}(G)$ the set of all walks.



Paths and walks





Walk kernel

Definition

- Let $\mathcal{S}_n$ denote the set of all possible label sequences of walks of length n (including vertices and edges labels), and $\mathcal{S} = \cup_{n\geq 1}\mathcal{S}_n$.
- For any graph $\mathcal{G}$ let a weight $\lambda_G(w)$ be associated to each walk $w \in \mathcal{W}(G)$.
- Let the feature vector $\Phi(G) = (\Phi_s(G))_{s\in\mathcal{S}}$ be defined by:

$$\Phi_s(G) = \sum_{w\in\mathcal{W}(G)} \lambda_G(w)\mathbf{1}\left(s \text{ is the label sequence of } w\right) .$$

- A walk kernel is a graph kernel defined by:

$$K_{walk}(G_1, G_2) = \sum_{s\in\mathcal{S}} \Phi_s(G_1)\Phi_s(G_2) .$$

Walk kernel

Definition

- Let $\mathcal{S}_n$ denote the set of all possible label sequences of walks of length n (including vertices and edges labels), and $\mathcal{S} = \cup_{n\geq 1}\mathcal{S}_n$.
- For any graph $\mathcal{G}$ let a weight $\lambda_G(w)$ be associated to each walk $w \in \mathcal{W}(G)$.
- Let the feature vector $\Phi(G) = \left(\Phi_s(G)\right)_{s\in\mathcal{S}}$ be defined by:

$$\Phi_s(G) = \sum_{w\in\mathcal{W}(G)} \lambda_G(w)\mathbf{1}\left(s \text{ is the label sequence of } w\right) .$$

- A walk kernel is a graph kernel defined by:

$$K_{walk}(G_1, G_2) = \sum_{s\in\mathcal{S}} \Phi_s(G_1)\Phi_s(G_2) .$$

Walk kernel examples

Examples

- The *n*th-order walk kernel is the walk kernel with $\lambda_G(w) = 1$ if the length of w is n, 0 otherwise. It compares two graphs through their common walks of length n.
- The random walk kernel is obtained with $\lambda_G(w) = P_G(w)$, where P_G is a Markov random walk on G. In that case we have:

$$K(G_1, G_2) = P(label(W_1) = label(W_2)) ,$$

  where W_1 and W_2 are two independant random walks on G_1 and G_2, respectively.
- The geometric walk kernel is obtained (when it converges) with $\lambda_G(w) = \beta^{length(w)}$, for $\beta > 0$. In that case the feature space is of infinite dimension.

Walk kernel examples

Examples

- The *n*th-order walk kernel is the walk kernel with $\lambda_G(w) = 1$ if the length of w is n, 0 otherwise. It compares two graphs through their common walks of length n.
- The random walk kernel is obtained with $\lambda_G(w) = P_G(w)$, where P_G is a Markov random walk on G. In that case we have:

$$K(G_1, G_2) = P(label(W_1) = label(W_2)),$$

  where W_1 and W_2 are two independant random walks on G_1 and G_2, respectively.
- The geometric walk kernel is obtained (when it converges) with $\lambda_G(w) = \beta^{length(w)}$, for $\beta > 0$. In that case the feature space is of infinite dimension.

Walk kernel examples

Examples

- The *n*th-order walk kernel is the walk kernel with $\lambda_G(w) = 1$ if the length of w is n, 0 otherwise. It compares two graphs through their common walks of length n.
- The random walk kernel is obtained with $\lambda_G(w) = P_G(w)$, where P_G is a Markov random walk on G. In that case we have:

  $$K(G_1, G_2) = P(label(W_1) = label(W_2)) ,$$

  where W_1 and W_2 are two independant random walks on G_1 and G_2, respectively.
- The geometric walk kernel is obtained (when it converges) with $\lambda_G(w) = \beta^{length(w)}$, for $\beta > 0$. In that case the feature space is of infinite dimension.

Computation of walk kernels

Proposition

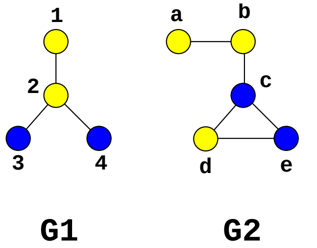
These three kernels (nth-order, random and geometric walk kernels) can be computed efficiently in polynomial time.

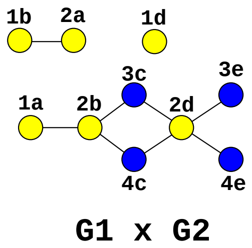
Product graph

Definition

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs with labeled vertices. The product graph $G = G_1 \times G_2$ is the graph $G = (V, E)$ with:

1. $V = \{(v_1, v_2) \in V_1 \times V_2 \ : \ v_1 \text{ and } v_2 \text{ have the same label}\}$,
2. $E = \left\{\left((v_1, v_2), (v_1', v_2')\right) \in V \times V \ : \ (v_1, v_1') \in E_1 \text{ and } (v_2, v_2') \in E_2\right\}$.





Walk kernel and product graph

Lemma

There is a bijection between:

1. The pairs of walks $w_1 \in \mathcal{W}_n(G_1)$ and $w_2 \in \mathcal{W}_n(G_2)$ with the same label sequences,
2. The walks on the product graph $w \in \mathcal{W}_n(G_1 \times G_2)$.

Corollary

$$\begin{aligned} K_{walk}(G_1, G_2) &= \sum_{s \in \mathcal{S}} \Phi_s(G_1)\Phi_s(G_2) \\ &= \sum_{(w_1,w_2) \in \mathcal{W}(G_1) \times \mathcal{W}(G_1)} \lambda_{G_1}(w_1)\lambda_{G_2}(w_2)\mathbf{1}(l(w_1) = l(w_2)) \\ &= \sum_{w \in \mathcal{W}(G_1 \times G_2)} \lambda_{G_1 \times G_2}(w) \,. \end{aligned}$$

Computation of the *n*th-order walk kernel

- For the *n*th-order walk kernel we have $\lambda_{G_1 \times G_2}(w) = 1$ if the length of w is n, 0 otherwise.
- Therefore:

$$K_{nth-order}(G_1, G_2) = \sum_{w \in \mathcal{W}_n(G_1 \times G_2)} 1 \,.$$

- Let A be the adjacency matrix of $G_1 \times G_2$. Then we get:

$$K_{nth-order}(G_1, G_2) = \sum_{i,j} [A^n]_{i,j} = \mathbf{1}^\top A^n \mathbf{1} \,.$$

- Computation in $O(n|G_1||G_2|d_1 d_2)$, where d_i is the maximum degree of G_i.

Computation of random and geometric walk kernels

- In both cases $\lambda_G(w)$ for a walk $w = v_1 \dots v_n$ can be decomposed as:

$$\lambda_G(v_1 \dots v_n) = \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i) \,.$$

- Let Λ_i be the vector of $\lambda^i(v)$ and Λ_t be the matrix of $\lambda^t(v, v')$:

$$\begin{aligned} K_{walk}(G_1, G_2) &= \sum_{n=1}^{\infty} \sum_{w \in \mathcal{W}_n(G_1 \times G_2)} \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i) \\ &= \sum_{n=0}^{\infty} \Lambda_i \Lambda_t^n \mathbf{1} \\ &= \Lambda_i \left(I - \Lambda_t\right)^{-1} \mathbf{1} \end{aligned}$$

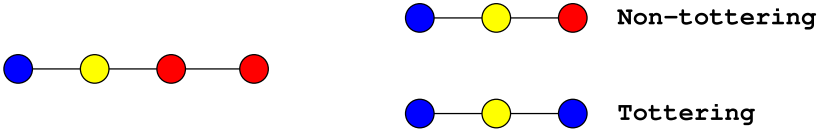
- Computation in $O(|G_1|^3 |G_2|^3)$

Outline

Extension 1: Non-tottering walk kernel

Tottering walks

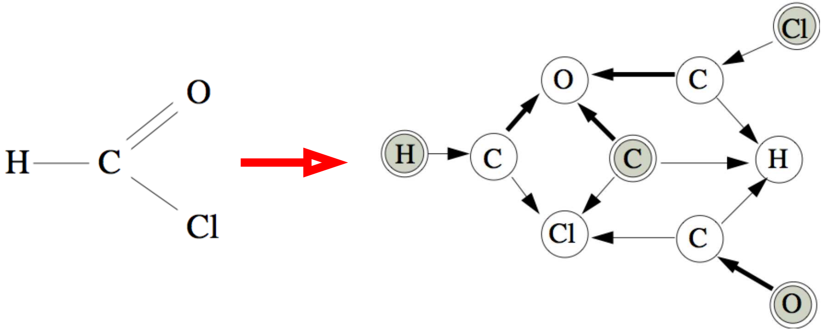
A tottering walk is a walk $w = v_1 \dots v_n$ with $v_i = v_{i+2}$ for some i.



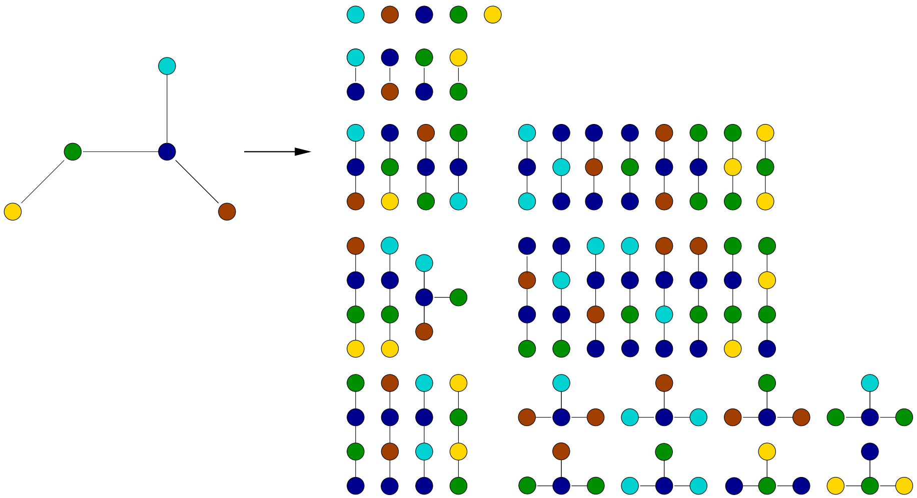
- Tottering walks seem irrelevant for many applications
- Focusing on non-tottering walks is a way to get closer to the path kernel (e.g., equivalent on trees).

Computation of the non-tottering walk kernel

- Second-order Markov random walk to prevent tottering walks
- Written as a first-order Markov random walk on an augmented graph
- Normal walk kernel on the augmented graph (which is always a directed graph).



Extension 2: Subtree kernels



Computation of the subtree kernel

- Like the walk kernel, amounts to compute the (weighted) number of subtrees in the product graph.
- Recursion: if $\mathcal{T}(v, n)$ denotes the weighted number of subtrees of depth n rooted at the vertex v, then:

$$\mathcal{T}(v, n+1) = \sum_{R \subset \mathcal{N}(v)} \prod_{v' \in R} \lambda_t(v, v') \mathcal{T}(v', n),$$

  where $\mathcal{N}(v)$ is the set of neighbors of v.
- Can be combined with the non-tottering graph transformation as preprocessing to obtain the non-tottering subtree kernel.

Outline

Chemoinformatics (Mahé et al., 2004)

MUTAG dataset

- aromatic/hetero-aromatic compounds
- high mutagenic activity /no mutagenic activity, assayed in *Salmonella typhimurium*.
- 188 compouunds: 125 + / 63 -

Results

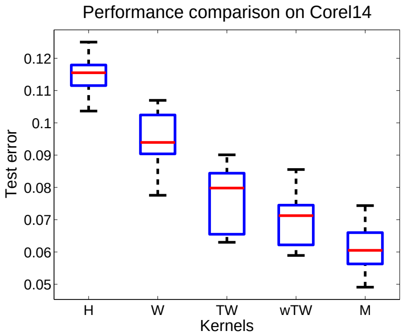
10-fold cross-validation accuracy

Method	Accuracy
Progol1	81.4%
2D kernel	91.2%

Image classification (Harchaoui and Bach, 2007)

COREL14 dataset

- 1400 natural images in 14 classes
- Compare kernel between histograms (H), walk kernel (W), subtree kernel (TW), weighted subtree kernel (wTW), and a combination (M).

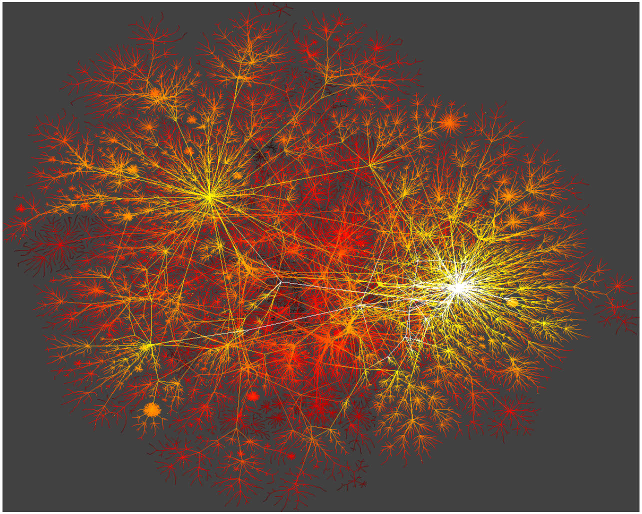


Part 3

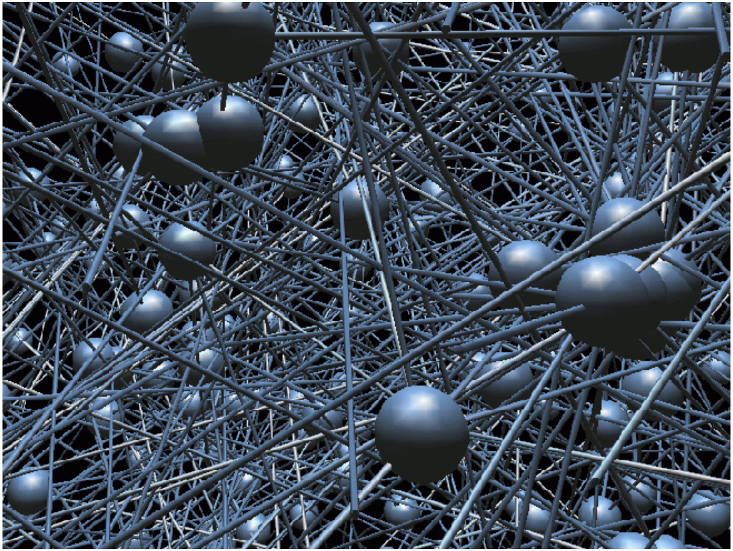
Kernels on Graphs

Outline

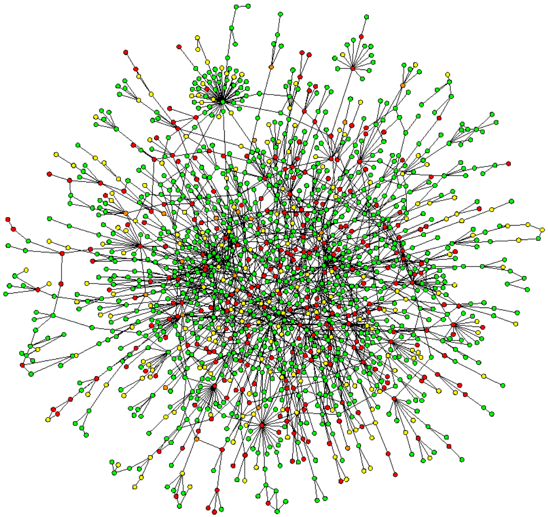
Example: web



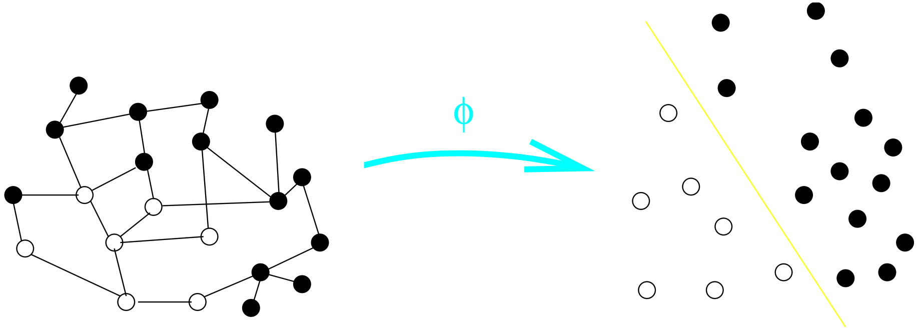
Example: social network



Example: protein-protein interaction



Kernel on a graph



- We need a kernel $K(\mathbf{x}, \mathbf{x}')$ between nodes of the graph.
- Example: predict gene protein functions from high-throughput protein-protein interaction data.

General remarks

Strategies to make a kernel on a graph

- $\mathcal{X}$ being finite, any symmetric semi-definite matrix K defines a valid p.d. kernel on $\mathcal{X}$.
- How to "translate" the graph topology into the kernel?
  - Direct geometric approach: $K_{i,j}$ should be "large" when $\mathbf{x}_i$ and $\mathbf{x}_j$ are "close" to each other on the graph?
  - Functional approach: $\| f \|_K$ should be "small" when f is "smooth" on the graph?
  - Link discrete/continuous: is there an equivalent to the continuous Gaussien kernel on the graph (e.g., limit by fine discretization)?

General remarks

Strategies to make a kernel on a graph

- $\mathcal{X}$ being finite, any symmetric semi-definite matrix K defines a valid p.d. kernel on $\mathcal{X}$.
- How to "translate" the graph topology into the kernel?
  - Direct geometric approach: $K_{i,j}$ should be "large" when $\mathbf{x}_i$ and $\mathbf{x}_j$ are "close" to each other on the graph?
  - Functional approach: $\| f \|_K$ should be "small" when f is "smooth" on the graph?
  - Link discrete/continuous: is there an equivalent to the continuous Gaussien kernel on the graph (e.g., limit by fine discretization)?

Outline

Conditionally p.d. kernels

Hilbert distance

- Any p.d. kernels is an inner product in a Hilbert space

$$K\left(\mathbf{x},\mathbf{x}'\right) = \left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right)\right\rangle_{\mathcal{H}} .$$

- It defines a Hilbert distance:

$$d_K\left(\mathbf{x},\mathbf{x}'\right)^2 = K\left(\mathbf{x},\mathbf{x}\right) + K\left(\mathbf{x}',\mathbf{x}'\right) - 2K\left(\mathbf{x},\mathbf{x}'\right)$$

- $-d_K^2$ is conditionally positive definite, i.e.:

$$\forall t > 0 , \quad \exp\left(-t d_K\left(\mathbf{x},\mathbf{x}'\right)^2\right) \text{ is p.d.}$$

Graph distance

Graph embedding in a Hilbert space

- Given a graph $G = (V, E)$, the graph distance $d_G(x, x')$ between any two vertices is the length of the shortest path between x and x'.
- We say that the graph $G = (V, E)$ can be embedded (exactly) in a Hilbert space if $-d_G$ is c.p.d., which implies in particular that $\exp(-td_G(x, x'))$ is p.d. for all $t > 0$.

Lemma

- *In general graphs can not be embedded exactly in Hilbert spaces.*
- *In some cases exact embeddings exists, e.g.:*
  - *trees can be embedded exactly,*
  - *closed chains can be embedded exactly.*

Graph distance

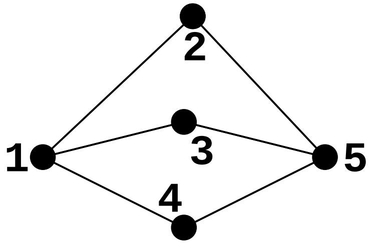
Graph embedding in a Hilbert space

- Given a graph $G = (V, E)$, the graph distance $d_G(x, x')$ between any two vertices is the length of the shortest path between x and x'.
- We say that the graph $G = (V, E)$ can be embedded (exactly) in a Hilbert space if $-d_G$ is c.p.d., which implies in particular that $\exp(-td_G(x, x'))$ is p.d. for all $t > 0$.

Lemma

- *In general graphs can not be embedded exactly in Hilbert spaces.*
- *In some cases exact embeddings exists, e.g.:*
  - *trees can be embedded exactly,*
  - *closed chains can be embedded exactly.*

Example: non-c.p.d. graph distance



$$d_G = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 1 & 0 & 2 & 2 & 1 \\ 1 & 2 & 0 & 2 & 1 \\ 1 & 2 & 2 & 0 & 1 \\ 2 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\lambda_{\min}\left(\left[e^{(-0.2 d_G(i,j))}\right]\right) = -0.028 < 0\,.$$

Graph distance on trees are c.p.d.

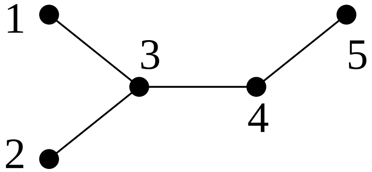
Proof

- Let $G = (V, E)$ a tree
- Fix a root $x_0 \in V$
- Represent any vertex $x \in V$ by a vector $\Phi(x) \in \mathbb{R}^{|E|}$, where $\Phi(x)_i = 1$ is the i-th edge is in the (unique) path between x and x_0, 0 otherwise.
- Then:

$$d_G(x, x') = \| \Phi(x) - \Phi(x') \|^2 ,$$

  and therefore $-d_G$ is c.p.d., in particular $\exp(-t d_G(x, x'))$ is p.d. for all $t > 0$.

Example

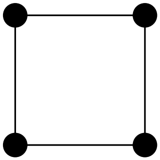


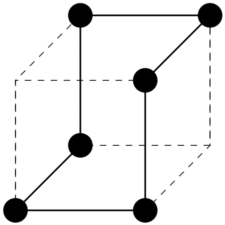
$$\left[e^{-d_G(i,j)} \right] = \begin{pmatrix} 1 & 0.14 & 0.37 & 0.14 & 0.05 \\ 0.14 & 1 & 0.37 & 0.14 & 0.05 \\ 0.37 & 0.37 & 1 & 0.37 & 0.14 \\ 0.14 & 0.14 & 0.37 & 1 & 0.37 \\ 0.05 & 0.05 & 0.14 & 0.37 & 1 \end{pmatrix}$$

Graph distance on closed chains are c.p.d.

Proof: case $|V| = 2p$

- Let $G = (V, E)$ a cycle with an even number of vertices $|V| = 2p$
- Fix a root $x_0 \in V$, number the $2p$ edges from x_0 to x_0.
- Map the $2p$ edges in $\mathbb{R}^p$ to $(e_1, \ldots, e_p, -e_1, \ldots, -e_p)$
- Map a vertex v to the sum of the edges in the shortest path between x_0 and v.





Outline

Functional approach

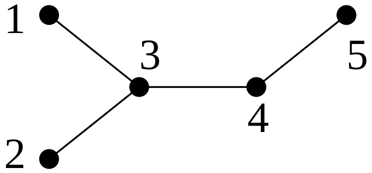
Motivation

- How to make p.d. kernel on general graphs?
- Making a kernel is equivalent to defining a RKHS.
- There are intuitive notions of smoothness on a graph

Idea

- Define a priori a smoothness functional on the functions $f : \mathcal{X} \to \mathbb{R}$.
- Show that it defines a RKHS and identify the corresponding kernel

Notations

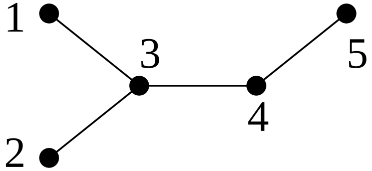


$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \qquad D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Graph Laplacian

Definition

The Laplacian of the graph is the matrix $L = D - A$.



$$L = A - D = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

Properties of the Laplacian

Lemma

Let $L = D - A$ be the Laplacian of a connected graph:

- *For any $f : \mathcal{X} \to \mathbb{R}$,*

$$\Omega(f) := \sum_{i \sim j} \left(f(\mathbf{x}_i) - f(\mathbf{x}_j) \right)^2 = f^\top L f$$

- *L is a symmetric positive semi-definite matrix*
- *0 is an eigenvalue with multiplicity 1 associated to the constant eigenvector $\mathbf{1} = (1, \ldots, 1)$*
- *The image of L is*

$$Im(L) = \left\{ f \in \mathbb{R}^m : \sum_{i=1}^{m} f_i = 0 \right\}$$

Proof: link between $\Omega(f)$ and L

$$
\begin{aligned}
\Omega(f) &= \sum_{i \sim j} \left(f(\mathbf{x}_i) - f(\mathbf{x}_j) \right)^2 \\
&= \sum_{i \sim j} \left(f(\mathbf{x}_i)^2 + f(\mathbf{x}_j)^2 - 2 f(\mathbf{x}_i) f(\mathbf{x}_j) \right) \\
&= \sum_{i=1}^{m} D_{i,i} f(\mathbf{x}_i)^2 - 2 \sum_{i \sim j} f(\mathbf{x}_i) f(\mathbf{x}_j) \\
&= f^\top D f - f^\top A f \\
&= f^\top L f
\end{aligned}
$$

Proof: eigenstructure of L

- L is symmetric because A and D are symmetric.
- For any $f \in \mathbb{R}^m$, $f^\top L f = \Omega(f) \geq 0$, therefore the (real-valued) eigenvalues of L are ≥ 0 : L is therefore positive semi-definite.
- f is an eigenvector associated to eigenvalue 0
  iff $f^\top L f = 0$
  iff $\sum_{i \sim j} \left(f(\mathbf{x}_i) - f(\mathbf{x}_j) \right)^2 = 0$,
  iff $f(\mathbf{x}_i) = f(\mathbf{x}_j)$ when $i \sim j$,
  iff f is constant (because the graph is connected).
- L being symmetric, $Im(L)$ is the orthogonal supplement of $Ker(L)$, that is, the set of functions orthogonal to $\mathbf{1}$. $\square$

Our first graph kernel

Theorem

The set $\mathcal{H} = \left\{ f \in \mathbb{R}^m : \sum_{i=1}^m f_i = 0 \right\}$ *endowed with the norm:*

$$\Omega(f) = \sum_{i \sim j} \left(f(\mathbf{x}_i) - f(\mathbf{x}_j) \right)^2$$

is a RKHS whose reproducing kernel is L^*, the pseudo-inverse of the graph Laplacian.

Proof (1/2)

- Resticted to $\mathcal{H}$, the symmetric bilinear form:

$$\langle f, g \rangle = f^\top L g$$

  is positive definite (because L is positive semi-definite, and $\mathcal{H} = Im(L)$). It is therefore a scalar product, making of $\mathcal{H}$ a Hilbert space (in fact Euclidean).
- The norm in this Hilbert space $\mathcal{H}$ is:

$$\| f \|^2 = \langle f, f \rangle = f^\top L f = \Omega(f) .$$

Proof (2/2)

To check that $\mathcal{H}$ is a RKHS with reproducing kernel $K = L^*$, it suffices to show that:

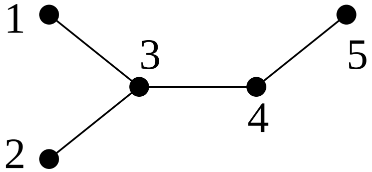
$$\begin{cases} \forall \mathbf{x} \in \mathcal{X}, & K_{\mathbf{x}} \in \mathcal{H}\,, \\ \forall \left(\mathbf{x}, f\right) \in \mathcal{X} \times \mathcal{H}, & \langle f, K_{\mathbf{x}} \rangle = f\left(\mathbf{x}\right)\,. \end{cases}$$

- $Ker(K) = Ker\left(L^*\right) = Ker\left(L\right)$, implying $K\mathbf{1} = 0$. Therefore, each row/column of K is in $\mathcal{H}$.
- For any $f \in \mathcal{H}$, if we note $g_i = \langle K(i, \cdot), f \rangle$ we get:

$$g = KLf = L^*Lf = \Pi_{\mathcal{H}}(f) = f\,.$$

As a conclusion $K = L^*$ is the reproducing kernel of $\mathcal{H}$. $\square$

Example



$$L^* = \begin{pmatrix} 0.88 & -0.12 & 0.08 & -0.32 & -0.52 \\ -0.12 & 0.88 & 0.08 & -0.32 & -0.52 \\ 0.08 & 0.08 & 0.28 & -0.12 & -0.32 \\ -0.32 & -0.32 & -0.12 & 0.48 & 0.28 \\ -0.52 & -0.52 & -0.32 & 0.28 & 1.08 \end{pmatrix}$$

Outline

The diffusion equation

Lemma

For any $\mathbf{x}_0 \in \mathbb{R}^d$*, the function:*

$$K_{\mathbf{x}_0}(\mathbf{x}, t) = K_t(\mathbf{x}_0, \mathbf{x}) = \frac{1}{(4\pi t)^{\frac{d}{2}}} \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}_0 \|^2}{4t}\right).$$

is solution of the diffusion equation:

$$\frac{\partial}{\partial t} K_{\mathbf{x}_0}(\mathbf{x}, t) = \Delta K_{\mathbf{x}_0}(\mathbf{x}, t).$$

with initial condition $K_{\mathbf{x}_0}(\mathbf{x}, 0) = \delta_{\mathbf{x}_0}(\mathbf{x})$.

Discrete diffusion equation

- For finite-dimensional $f_t \in \mathbb{R}^m$, the diffusion equation becomes:

$$\frac{\partial}{\partial t} f_t = -L f_t$$

which admits the following solution:

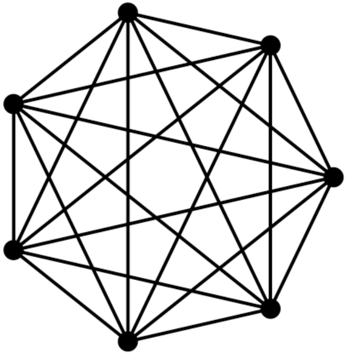
$$f_t = f_0 e^{-tL}$$

- This suggest to consider:

$$K = e^{-tL}$$

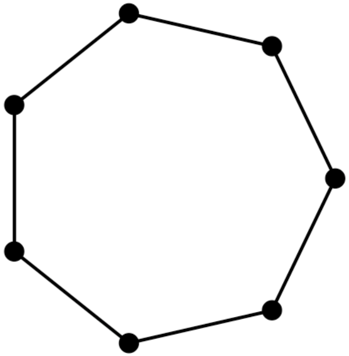
which is indeed symmetric positive semi-definite. We call it the diffusion kernel or heat kernel.

Example: complete graph



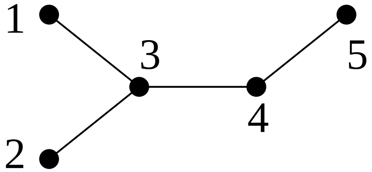
$$K_{i,j} = \begin{cases} \frac{1+(m-1)e^{-tm}}{m} & \text{for } i = j, \\ \frac{1-e^{-tm}}{m} & \text{for } i \neq j. \end{cases}$$

Example: closed chain



$$K_{i,j} = \frac{1}{m} \sum_{\nu=0}^{m-1} \exp\left[-2t\left(1 - \cos\frac{2\pi\nu}{m}\right)\right] \cos\frac{2\pi\nu(i-j)}{m}.$$

Example



$$e^{-L} = \begin{pmatrix} 0.50 & 0.13 & 0.24 & 0.10 & 0.04 \\ 0.13 & 0.50 & 0.24 & 0.10 & 0.04 \\ 0.24 & 0.24 & 0.24 & 0.18 & 0.10 \\ 0.10 & 0.10 & 0.18 & 0.32 & 0.30 \\ 0.04 & 0.04 & 0.10 & 0.30 & 0.52 \end{pmatrix}$$

Outline

Spectrum of the diffusion kernel

- Let $0 = \lambda_1 < \lambda_2 \leq \ldots \leq \lambda_m$ be the eigenvalues of the Laplacian:

$$L = \sum_{i=1}^{m} \lambda_i u_i u_i^\top \quad (\lambda_i \geq 0)$$

- The diffusion kernel K_t is an invertible matrix because its eigenvalues are strictly positive:

$$K_t = \sum_{i=1}^{m} e^{-t\lambda_i} u_i u_i^\top$$

Norm in the diffusion RKHS

- For any function $f \in \mathbb{R}^m$, let:

$$\hat{f}_i = u_i^\top f$$

  be the Fourier coefficients of f (projection of f onto the eigenbasis of K).
- The RKHS norm of f is then:

$$\| f \|_{K_t}^2 = f^\top K^{-1} f = \sum_{i=1}^{m} e^{t\lambda_i} \hat{f}_i^2 .$$

Generalization

This observation suggests to define a whole family of kernels:

$$K_r = \sum_{i=1}^{m} r(\lambda_i) u_i u_i^\top$$

associated with the following RKHS norms:

$$\| f \|_{K_r}^2 = \sum_{i=1}^{m} \frac{\hat{f}_i^2}{r(\lambda_i)}$$

where $r : \mathbb{R}^+ \to \mathbb{R}_*^+$ is a non-increasing function.

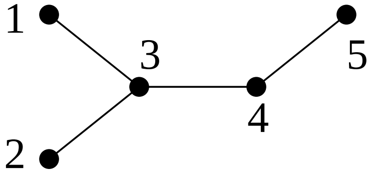
Example : regularized Laplacian

$$r(\lambda) = \frac{1}{\lambda + \epsilon}, \qquad \epsilon > 0$$

$$K = \sum_{i=1}^{m} \frac{1}{\lambda_i + \epsilon} u_i u_i^\top = (L + \epsilon I)^{-1}$$

$$\| f \|_K^2 = f^\top K^{-1} f = \sum_{i \sim j} \left(f\left(\mathbf{x}_i\right) - f\left(\mathbf{x}_j\right)\right)^2 + \epsilon \sum_{i=1}^{m} f\left(\mathbf{x}_i\right)^2 .$$

Example



$$(L+I)^{-1} = \begin{pmatrix} 0.60 & 0.10 & 0.19 & 0.08 & 0.04 \\ 0.10 & 0.60 & 0.19 & 0.08 & 0.04 \\ 0.19 & 0.19 & 0.38 & 0.15 & 0.08 \\ 0.08 & 0.08 & 0.15 & 0.46 & 0.23 \\ 0.04 & 0.04 & 0.08 & 0.23 & 0.62 \end{pmatrix}$$

Outline

Applications 1: graph partitioning

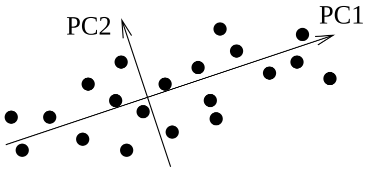
- A classical relaxation of graph partitioning is:

$$\min_{f \in \mathbb{R}^{\mathcal{X}}} \sum_{i \sim j} \left(f_i - f_j\right)^2 \quad \text{s.t.} \sum_i f_i^2 = 1$$

- This can be rewritten

$$\max_f \sum_i f_i^2 \text{ s.t. } \quad \| f \|_{\mathcal{H}} \leq 1$$

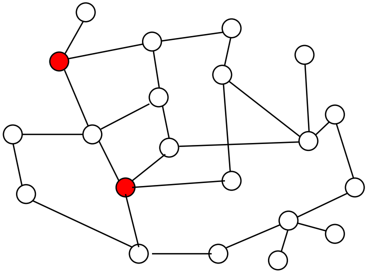
- This is principal component analysis in the RKHS (“kernel PCA”)

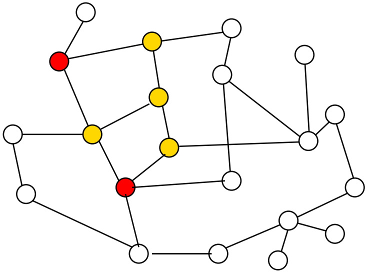


Applications 2: search on a graph

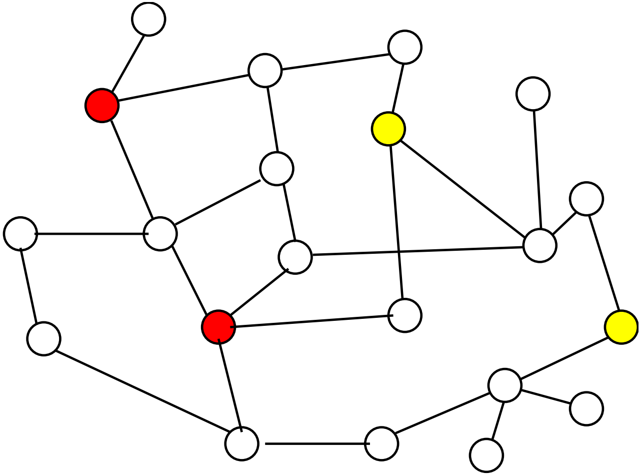
- Let $x_1, \ldots, x_q$ a set of q nodes (the query). How to find “similar” nodes (and rank them)?
- One solution:

$$\min_f \| f \|_{\mathcal{H}} \quad \text{s.t.} \quad f(x_i) \geq 1 \text{ for } i = 1, \ldots, q.$$

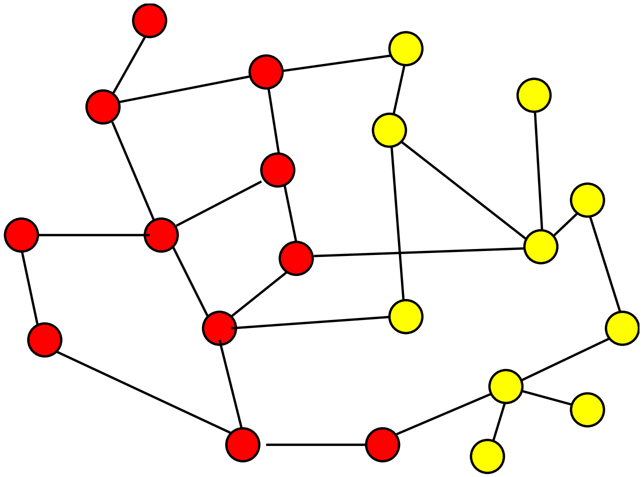




Application 3: Semi-supervised learning



Application 3: Semi-supervised learning



Application 4: Tumor classification from microarray data

Data available

- Gene expression measures for more than 10k genes
- Measured on less than 100 samples of two (or more) different classes (e.g., different tumors)

Goal

- Design a classifier to automatically assign a class to future samples from their expression profile
- Interpret biologically the differences between the classes

Application 4: Tumor classification from microarray data

Data available

- Gene expression measures for more than 10k genes
- Measured on less than 100 samples of two (or more) different classes (e.g., different tumors)

Goal

- Design a classifier to automatically assign a class to future samples from their expression profile
- Interpret biologically the differences between the classes

Linear classifiers

The approach

- Each sample is represented by a vector $x = (x_1, \ldots, x_p)$ where $p > 10^5$ is the number of probes
- Classification: given the set of labeled sample, learn a linear decision function:

$$f(x) = \sum_{i=1}^{p} \beta_i x_i + \beta_0 \, ,$$

- Interpretation: the weight β_i quantifies the influence of gene i for the classification

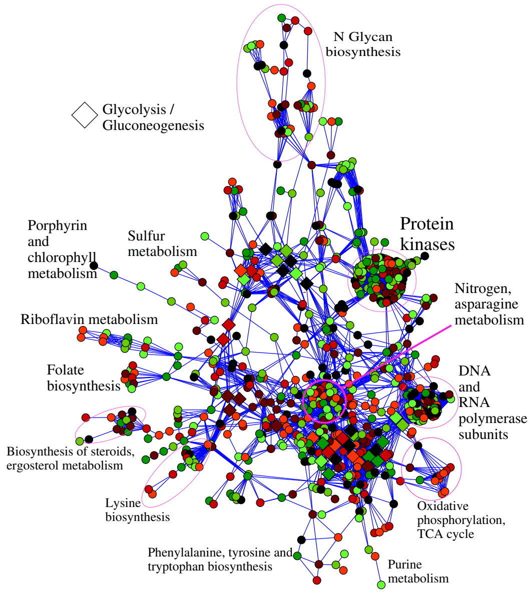
Pitfalls

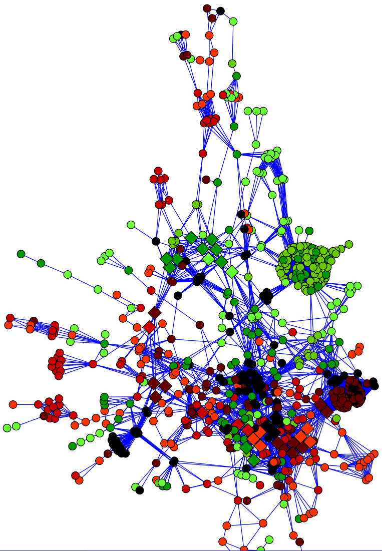
- No robust estimation procedure exist for 100 samples in 10^5 dimensions!

Prior knowledge

- We know the functions of many genes, and how they interact together.
- This can be represented as a graph of genes, where connected genes perform some action together
- Prior knowledge: constraint the weights of genes that work together to be similar
- Mathematically: constrain the norm of the weight vector in the RKHS of the diffusion kernel.

Comparison


N Glycan biosynthesis
◇ Glycolysis / Gluconeogenesis
Porphyrin and chlorophyll metabolism
Sulfur metabolism
Protein kinases
Nitrogen, asparagine metabolism
Riboflavin metabolism
Folate biosynthesis
DNA and RNA polymerase subunits
Biosynthesis of steroids, ergosterol metabolism
Lysine biosynthesis
Oxidative phosphorylation, TCA cycle
Phenylalanine, tyrosine and tryptophan biosynthesis
Purine metabolism



Conclusion

Conclusion

What we saw

- Extension of machine learning algorithms to graph data through the definition of positive definite kernels for and on graphs
- A variety of solutions have been proposed, borrowing ideas from graph algorithms and spectral graph theory.
- Increasingly used in real-world applications.

Unanswered question

- Theoretical foundations to guide the choice of kernel?
- How to design / choose / learn a kernel for a given application in practice?
- How to improve scalability of kernel methods + graph kernels to large datasets?

Further reading

Kernels and RKHS: general

N. Aronszajn.
Theory of reproducing kernels.
Trans. Am. Math. Soc., 68:337 – 404, 1950.

C. Berg, J. P. R. Christensen, and P. Ressel.
Harmonic analysis on semigroups.
Springer-Verlag, New-York, 1984.

G. Wahba.
Spline Models for Observational Data, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*.
SIAM, Philadelphia, 1990.

Further reading

Learning with kernels

V. N. Vapnik.
Statistical Learning Theory.
Wiley, New-York, 1998.

B. Schölkopf and A. J. Smola.
Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.
MIT Press, Cambridge, MA, 2002.

J. Shawe-Taylor and N. Cristianini.
Kernel Methods for Pattern Analysis.
Cambridge University Press, 2004.

B. Schölkopf, K. Tsuda, and J.-P. Vert.
Kernel Methods in Computational Biology.
MIT Press, 2004.

Further reading

Kernels for graphs

T. Gärtner, P. Flach, and S. Wrobel.
On graph kernels: hardness results and efficient alternatives.
Proceedings of COLT, p.129–143, Springer, 2003.

H. Kashima, K. Tsuda, and A. Inokuchi.
Marginalized Kernels between Labeled Graphs.
Proceedings of ICML, p. 321–328. AAAI Press, 2003.

P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert.
Graph kernels for molecular structure-activity relationship analysis with support vector machines.
J Chem Inf Model, 45(4):939–51, 2005.

Z. Harchaoui and F. Bach.
Image classification with segmentation graph kernels.
Tech report N35/06/MM, Ecole des Mines de Paris, 2006.

Further reading

Kernels on graphs

R. I. Kondor and J. Lafferty.
Diffusion Kernels on Graphs and Other Discrete Input.
In *ICML 2002*, 2002.

J.-P. Vert and M. Kanehisa.
Graph-driven features extraction from microarray data using diffusion kernels and kernel CCA.
In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Adv. Neural Inform. Process. Syst.*, pages 1449–1456. MIT Press, 2003.

F. Rapaport, A. Zynoviev, M. Dutreix, E. Barillot, and J.-P. Vert.
Classification of microarray data using gene networks.
BMC Bioinformatics, 8:35, 2007.