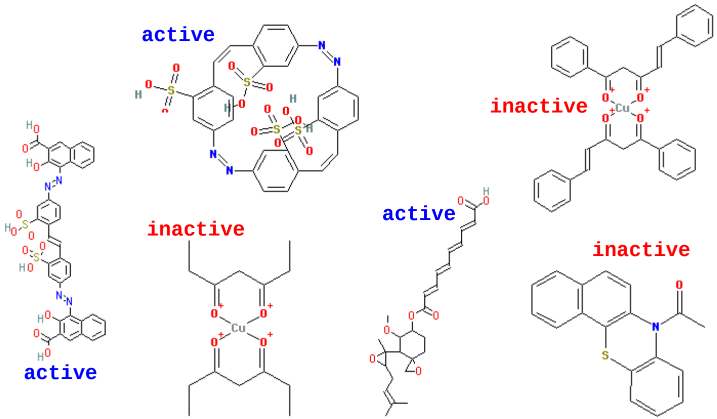
Virtual Screening with Support Vector Machines

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

Center for Computational Biology
Ecole des Mines de Paris, ParisTech

Multimodal Data Project Meeting, National Institute of Informatics, Tokyo, October 19th, 2007

Ligand-Based Virtual Screening and QSAR



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

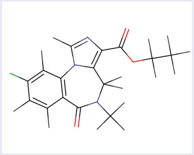
More formally...

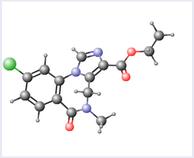
Objective

Build models to predict biochemical properties Y of small molecules from their structures X, using a training set of (X, Y) pairs.

Structures X

$C_{15}H_{14}ClN_3O_3$





Properties Y

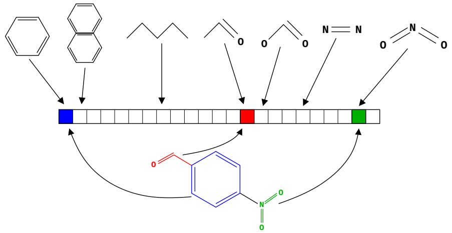
- binding to a therapeutic target,
- pharmacokinetics (ADME),
- toxicity...

Classical approaches

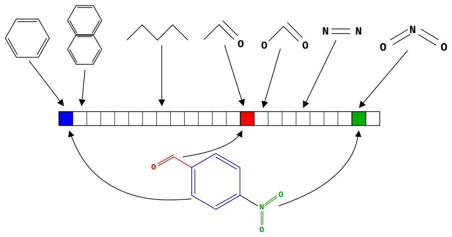
Two steps

1. Map each molecule to a vector of fixed dimension using molecular descriptors
   - Global properties of the molecules (mass, logP...)
   - 2D and 3D descriptors (substructures, fragments, ...)
2. Apply an algorithm for regression or pattern recognition.
   - PLS, ANN, ...

Example: 2D structural keys



Which descriptors?



Difficulties

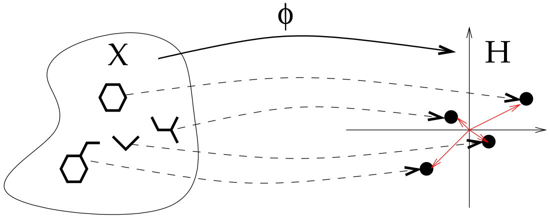
- Many descriptors are needed to characterize various features (in particular for 2D and 3D descriptors)
- But too many descriptors are harmful for memory storage, computation speed, statistical estimation

Kernels

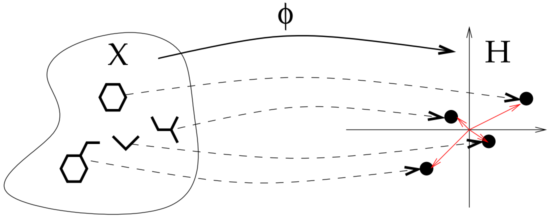
Definition

- Let $\Phi(x) = (\Phi_1(x), \ldots, \Phi_p(x))$ be a vector representation of the molecule x
- The kernel between two molecules is defined by:

$$K(x, x') = \Phi(x)^\top \Phi(x') = \sum_{i=1}^{p} \Phi_i(x)\Phi_i(x') .$$



The kernel trick



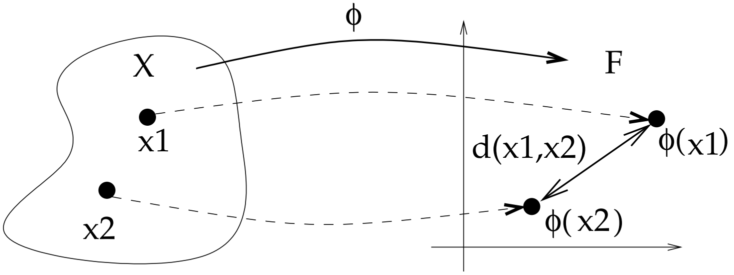
The trick

1. Computing the kernel $K(x, x')$ is often more efficient than computing $\Phi(x)$, especially in high or infinite dimensions! Ex:

$$K(x, x') = \exp\left(-\gamma \| x - x' \|^2\right) .$$

2. Many linear algorithms for regression or pattern recognition can be expressed only in terms of kernels.

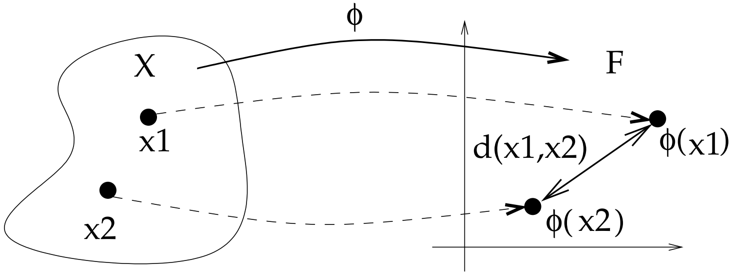
Kernel trick example: computing distances in the feature space



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

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

Kernel trick example: computing distances in the feature space



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

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

Other algorithms

Kernel methods

You don't like nearest-neighbor classification, or your problem is not binary classification, but you would like to benefit from the kernel trick (nonlinearity, structured data etc...)? Try other kernel methods that extend your favorite algorithm to handle kernels:

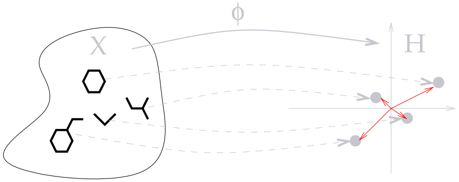
- Support Vector Machines,
- kernel PLS,
- kernel PCA,
- kriging,
- kernel perceptron,
- kernel logistic regression,
- and many more!

Making kernels for molecules

- Strategy 1: use well-known molecular descriptors to represent molecules m as vectors $\Phi(m)$, and then use kernels for vectors, e.g.:

$$K(m_1, m_2) = \Phi(m_1)^\top \Phi(m_2).$$

- Strategy 2: invent new kernels to do things you can not do with strategy 1, such as using an infinite number of descriptors. We will now see two examples of this strategy, extending 2D and 3D molecular descriptors.

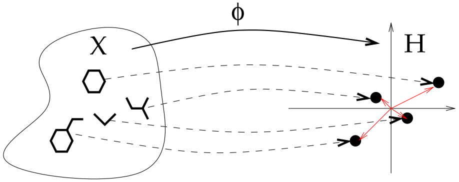


Making kernels for molecules

- Strategy 1: use well-known molecular descriptors to represent molecules m as vectors $\Phi(m)$, and then use kernels for vectors, e.g.:

$$K(m_1, m_2) = \Phi(m_1)^\top \Phi(m_2).$$

- Strategy 2: invent new kernels to do things you can not do with strategy 1, such as using an infinite number of descriptors. We will now see two examples of this strategy, extending 2D and 3D molecular descriptors.



Summary

The problem

- Regression and pattern recognition over molecules
- Classical vector representation is both statistically and computationally challenging

The kernel approach

By defining a kernel for molecules we can work implicitly in large (potentially infinite!) dimensions:

- Allows to consider a large number of potentially important features.
- No need to store explicitly the vectors (no problem of memory storage or hash clashes) thanks to the kernel trick
- Use of regularized statistical algorithm (SVM, kernel PLS, kernel perceptron...)to handle the statistical problem of large dimension

Summary

The problem

- Regression and pattern recognition over molecules
- Classical vector representation is both statistically and computationally challenging

The kernel approach

By defining a kernel for molecules we can work implicitly in large (potentially infinite!) dimensions:

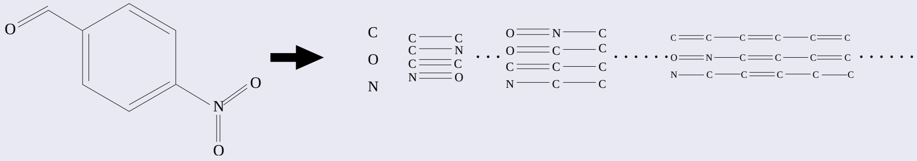
- Allows to consider a large number of potentially important features.
- No need to store explicitly the vectors (no problem of memory storage or hash clashes) thanks to the kernel trick
- Use of regularized statistical algorithm (SVM, kernel PLS, kernel perceptron...)to handle the statistical problem of large dimension

Outline

Motivation: 2D Fingerprints

Features

A vector indexed by a large set of molecular fragments



Pros

- Many features
- Easy to detect

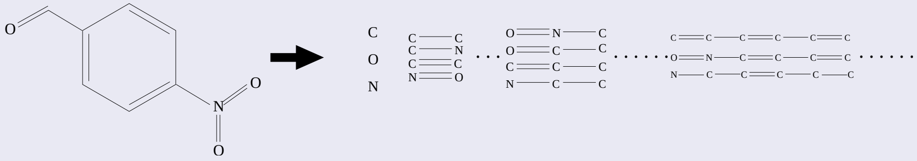
Cons

- Too many features?
- Hashing $\Longrightarrow$ clashes

Motivation: 2D Fingerprints

Features

A vector indexed by a large set of molecular fragments



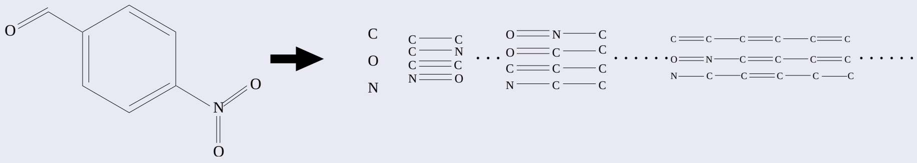
Pros

- Many features
- Easy to detect

Cons

- Too many features?
- Hashing $\Longrightarrow$ clashes

SVM approach



Let $\Phi(x)$ the vector of fragment counts:

- Long fragments lead to large dimensions :
  SVM can learn in high dimension
- $\Phi(x)$ is too long to be stored, and hashes induce clashes:
  SVM do not need $\Phi(x)$, they just need the kernel

$$K(x,x') = \phi(x)^\top \phi(x')\ .$$

2D fingerprint kernel

Definition

- For any $d > 0$ let $\phi_d(x)$ be the vector of counts of all fragments (walks) of length d:

$$\phi_1(x) = (\quad \texttt{\#(C),\#(O),\#(N),} \quad \ldots)^\top$$
$$\phi_2(x) = (\quad \texttt{\#(C-C),\#(C=O),\#(C-N),} \quad \ldots)^\top \quad \text{etc...}$$

- A 2D fingerprint walk kernel is defined, for a function $\lambda(d) \geq 0$, by

$$K_{2D}(x, x') = \sum_{d=1}^{\infty} \lambda(d) \phi_d(x)^\top \phi_d(x') \, .$$

- This is an inner product in the space of 2D fingerprints of infinite length.

2D Walk kernel examples

Examples

- The *n*th-order walk kernel is the walk kernel with $\lambda(n) = 1$ and $\lambda(d) = 0$ for $d \neq n$. It compares two graphs through their common walks of length n.
- The geometric walk kernel is obtained (when it converges) with $\lambda(d) = \beta^d$, for $\beta > 0$. In that case the feature space is of infinite dimension (Gärtner et al., 2003).
- Other variants are possible (e.g., random walk kernel of Kashima et al.)

2D Walk kernel examples

Examples

- The *n*th-order walk kernel is the walk kernel with $\lambda(n) = 1$ and $\lambda(d) = 0$ for $d \neq n$. It compares two graphs through their common walks of length n.
- The geometric walk kernel is obtained (when it converges) with $\lambda(d) = \beta^d$, for $\beta > 0$. In that case the feature space is of infinite dimension (Gärtner et al., 2003).
- Other variants are possible (e.g., random walk kernel of Kashima et al.)

2D Walk kernel examples

Examples

- The nth-order walk kernel is the walk kernel with $\lambda(n) = 1$ and $\lambda(d) = 0$ for $d \neq n$. It compares two graphs through their common walks of length n.
- The geometric walk kernel is obtained (when it converges) with $\lambda(d) = \beta^d$, for $\beta > 0$. In that case the feature space is of infinite dimension (Gärtner et al., 2003).
- Other variants are possible (e.g., random walk kernel of Kashima et al.)

2D kernel computation

Proposition

These 2D walk kernels can be computed efficiently in polynomial time.

Remarks

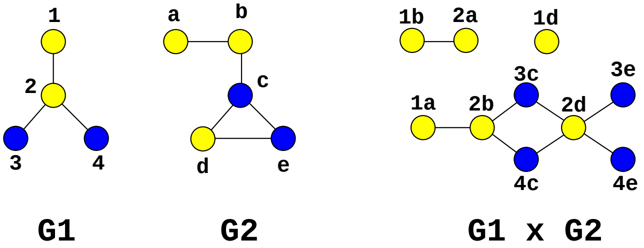
- The complexity is not always related to the length of the fragments considered (although faster computations are possible if the length is limited).
- Solves the problem of clashes and memory storage.
- Allows to work with infinite-length fingerprints without computing them!

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. $E = \{((v_1, v_2), (v_1', v_2')) \in V \times V \ : \ (v_1, v_1') \in E_1 \text{ and } (v_2, v_2') \in E_2\}$.

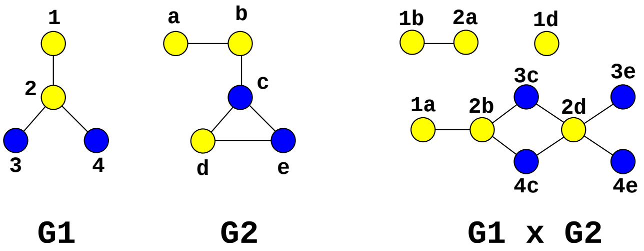


Walk kernel and product graph

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 $w \in \mathcal{W}_n(G_1 \times G_2)$.



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 nth-order walk kernel

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

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

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

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

- 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 $w = v_1 \ldots v_n$ can be decomposed as:

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

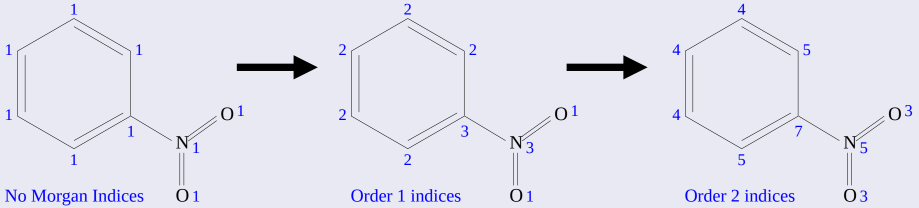
- Let Λ_i be the vector of $\lambda^i(v)$ and Λ_t be the matrix of $\lambda^t(v, v')$:

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

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

Extensions 1: label enrichment

Atom relabebling with the Morgan index

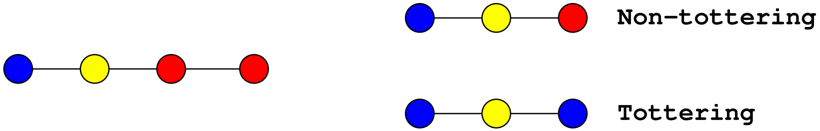


- Compromise between fingerprints and structural keys features.
- Other relabeling schemes are possible (graph coloring).
- Faster computation with more labels (less matches implies a smaller product graph).

Extension 2: Non-tottering walk kernel

Tottering walks

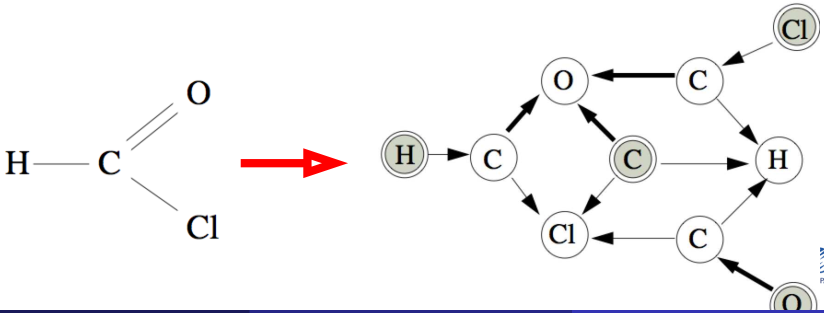
A tottering walk is a walk $w = v_1 \dots v_n$ with $v_i = v_{i+2}$ for some i.



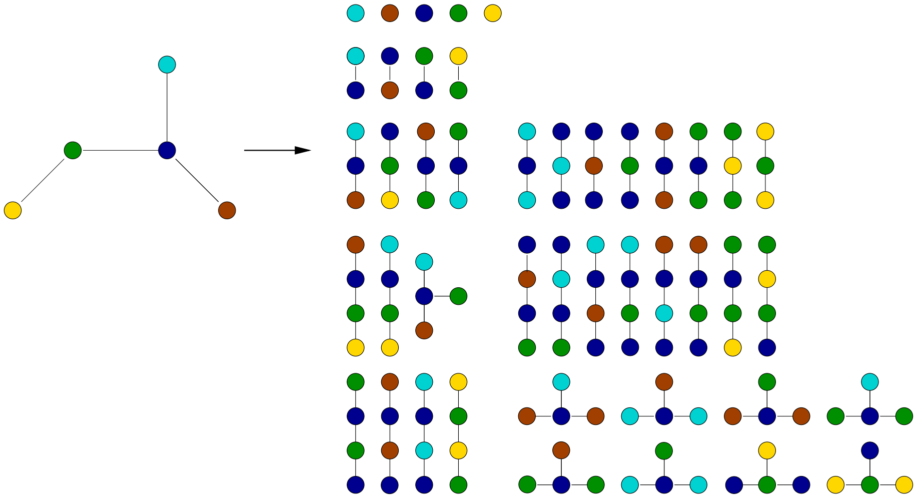
- Tottering walks seem irrelevant for many applications
- Focusing on non-tottering walks is a way to get closer to the path kernel (e.g., equivalent on trees).

Computation of the non-tottering walk kernel (Mahé et al., 2005)

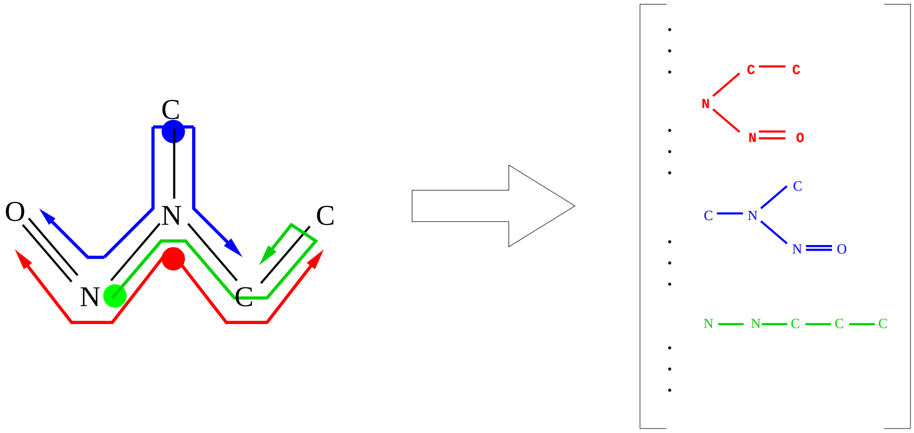
- Second-order Markov random walk to prevent tottering walks
- Written as a first-order Markov random walk on an augmented graph
- Normal walk kernel on the augmented graph (which is always a directed graph).



Extension 2: Subtree kernels



Example: Tree-like fragments of molecules



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.

Experiments

MUTAG dataset

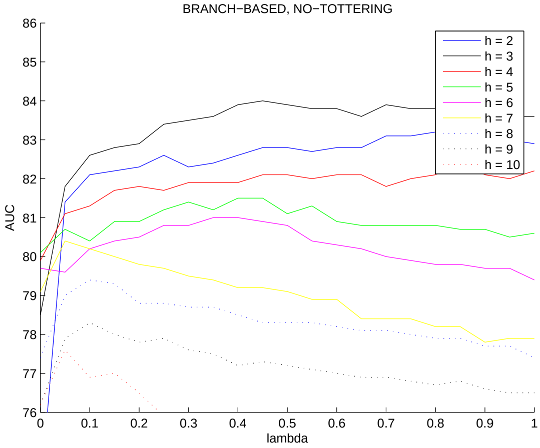
- aromatic/hetero-aromatic compounds
- high mutagenic activity /no mutagenic activity, assayed in *Salmonella typhimurium*.
- 188 compouunds: 125 + / 63 -

Results

10-fold cross-validation accuracy

Method	Accuracy
Progol1	81.4%
2D kernel	91.2%

Subtree kernels

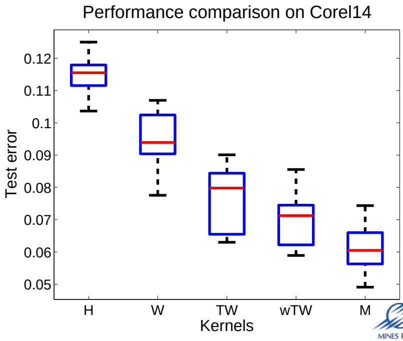


AUC as a function of the branching factors for different tree depths (from Mahé et al., 2007).

Image classification (Harchaoui and Bach, 2007)

COREL14 dataset

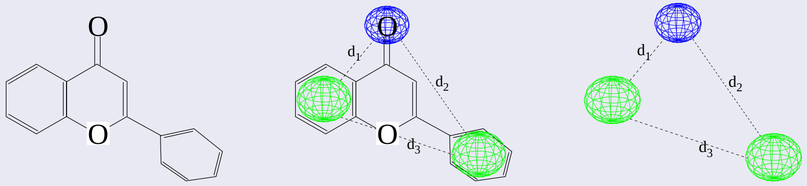
- 1400 natural images in 14 classes
- Compare kernel between histograms (H), walk kernel (W), subtree kernel (TW), weighted subtree kernel (wTW), and a combination (M).



Outline

Space of pharmacophore

3-points pharmacophores



A set of 3 atoms, and 3 inter-atom distances:

$$\mathcal{T} = \{((x_1, x_2, x_3), (d_1, d_2, d_3)), x_i \in \{\text{atom types}\}; d_i \in \mathbb{R}\}$$

3D fingerprint kernel

Pharmacophore fingerprint

1. Discretize the space of pharmacophores $\mathcal{T}$ (e.g., 6 atoms or groups of atoms, 6-7 distance bins) into a finite set $\mathcal{T}_d$
2. Count the number of occurrences $\phi_t(x)$ of each pharmacophore bin t in a given molecule x, to form a pharmacophore fingerprint.

3D kernel

A simple 3D kernel is the inner product of pharmacophore fingerprints:

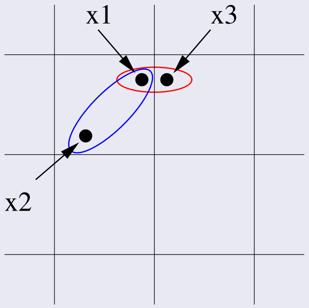
$$K(x, x') = \sum_{t \in \mathcal{T}_d} \phi_t(x)\phi_t(x') \, .$$

Discretization of the pharmacophore space

Common issues

1. If the bins are too large, then they are not specific enough
2. If the bins are too large, then they are too specific

In all cases, the arbitrary position of boundaries between bins affects the comparison:



$\rightarrow d(x_1, x_3) < d(x_1, x_2)$
BUT $\text{bin}(x_1) = \text{bin}(x_2) \neq \text{bin}(x_3)$

Kernels between pharmacophores

A small trick

$$
\begin{aligned}
K(x,y) &= \sum_{t \in \mathcal{T}_d} \phi_t(x)\phi_t(y) \\
&= \sum_{t \in \mathcal{T}_d} \Big(\sum_{p_x \in \mathcal{P}(x)} \mathbf{1}(\mathrm{bin}(\mathbf{p_x}) = \mathbf{t}) \Big) \Big(\sum_{p_y \in \mathcal{P}(y)} \mathbf{1}(\mathrm{bin}(\mathbf{p_y}) = \mathbf{t}) \Big) \\
&= \sum_{p_x \in \mathcal{P}(x)} \sum_{p_y \in \mathcal{P}(y)} \mathbf{1}(\mathrm{bin}(\mathbf{p_x}) = \mathrm{bin}(\mathbf{p_y}))
\end{aligned}
$$

General pharmacophore kernel

$$
K(x,y) = \sum_{p_x \in \mathcal{P}(x)} \sum_{p_y \in \mathcal{P}(y)} K_P(p_x, p_y)
$$

New pharmacophore kernels

- Discretizing the pharmacophore space is equivalent to taking the following kernel between individual pharmacophores:

$$K_P(p_1, p_2) = \mathbf{1}\left(\text{bin}(\mathbf{p_x}) = \text{bin}(\mathbf{p_y})\right)$$

- For general kernels, there is no need for discretization!
- For example, is $d(p_1, p_2)$ is a Euclidean distance between pharmacophores, take:

$$K_P\left(p_1, p_2\right) = \exp\left(-\gamma d\left(p_1, p_2\right)\right) \ .$$

Experiments

4 public datasets

- BZR: ligands for the benzodiazepine receptor
- COX: cyclooxygenase-2 inhibitors
- DHFR: dihydrofolate reductase inhibitors
- ER: estrogen receptor ligands

	TRAIN		TEST	
	Pos	Neg	Pos	Neg
BZR	94	87	63	62
COX	87	91	61	64
DHFR	84	149	42	118
ER	110	156	70	110

Experiments

Results (accuracy)

Kernel	BZR	COX	DHFR	ER
2D (Tanimoto)	71.2	63.0	76.9	77.1
3D fingerprint	75.4	67.0	76.9	78.6
3D not discretized	**76.4**	**69.8**	**81.9**	**79.8**

Outline

Summary

- SVM is a powerful and flexible machine learning algorithm. The kernel trick allows the manipulation of non-vectorial objects at the cost of defining a kernel function.
- The 2D kernel for molecule extends classical fingerprint-based approches. It solves the problem of bit clashes, allows infinite fingerprints and various extensions.
- The 3D kernel for molecule extends classical pharmacophore fingerprint-based approaches. It solves the problems of bit clashes and of discretization.
- Both kernels improve upon their classical counterparts, and provide competitive results on benchmark datasets.

Acknowledgements

- Pierre Mahé (CBIO)
- Tatsuya Akutsu, Nobuhisa Ueda, Jean-Luc Perret (Kyoto University)
- Liva Ralaivola (U Marseille)

References

- Kashima, H., Tsuda, K., and Inokuchi, A. *Marginalized kernels between labeled graphs*. Proceedings of the 20th ICML, 2003, pp. 321-328.
- P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. *Graph kernels for molecular structure-activity relationship analysis with SVM*. J. Chem. Inf. Model., 45(4):939-951, 2005.
- P. Mahé, L. Ralaivola, V. Stoven, and J-P Vert. *The pharmacophore kernel for virtual screening with SVM*. J. Chem. Inf. Model., 46(5):2003-2014, 2006.
- P. Mahé and J.-P. Vert. *Graph kernels based on tree patterns for molecules*. Technical report HAL:ccsd-00095488, 2006.
- P. Mahé. *Kernel design for virtual screening of small molecules with support vector machines*. PhD thesis, Ecole des Mines de Paris, 2006.
- Open-source kernels for chemoinformatics: `http://chemcpp.sourceforge.net/`

MINES PARIS ARMINES