Some contributions of machine learning to bioinformatics

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

Mines ParisTech / Institut Curie / Inserm

Ecole Polytechnique, Palaiseau, France, Feb 10, 2008.

Where I come from

institutCurie

MINES PARIS ParisTech

Inserm

- A joint lab about “Cancer computational genomics, bioinformatics, biostatistics and epidemiology”
- Located in th Institut Curie, a major hospital and cancer research institute in Europe

”Statistical machine learning for cancer informatics” team

Main topics

- Towards better diagnosis, prognosis, and personalized medicine
  - Supervised classification of genomic, transcriptomic, proteomic data; heterogeneous data integration
- Towards new drug targets
  - Systems biology, reconstruction of gene networks, pathway enrichment analysis, multidimensional phenotyping of cell populations.
- Towards new drugs
  - Ligand-based virtual screening, *in silico* chemogenomics.

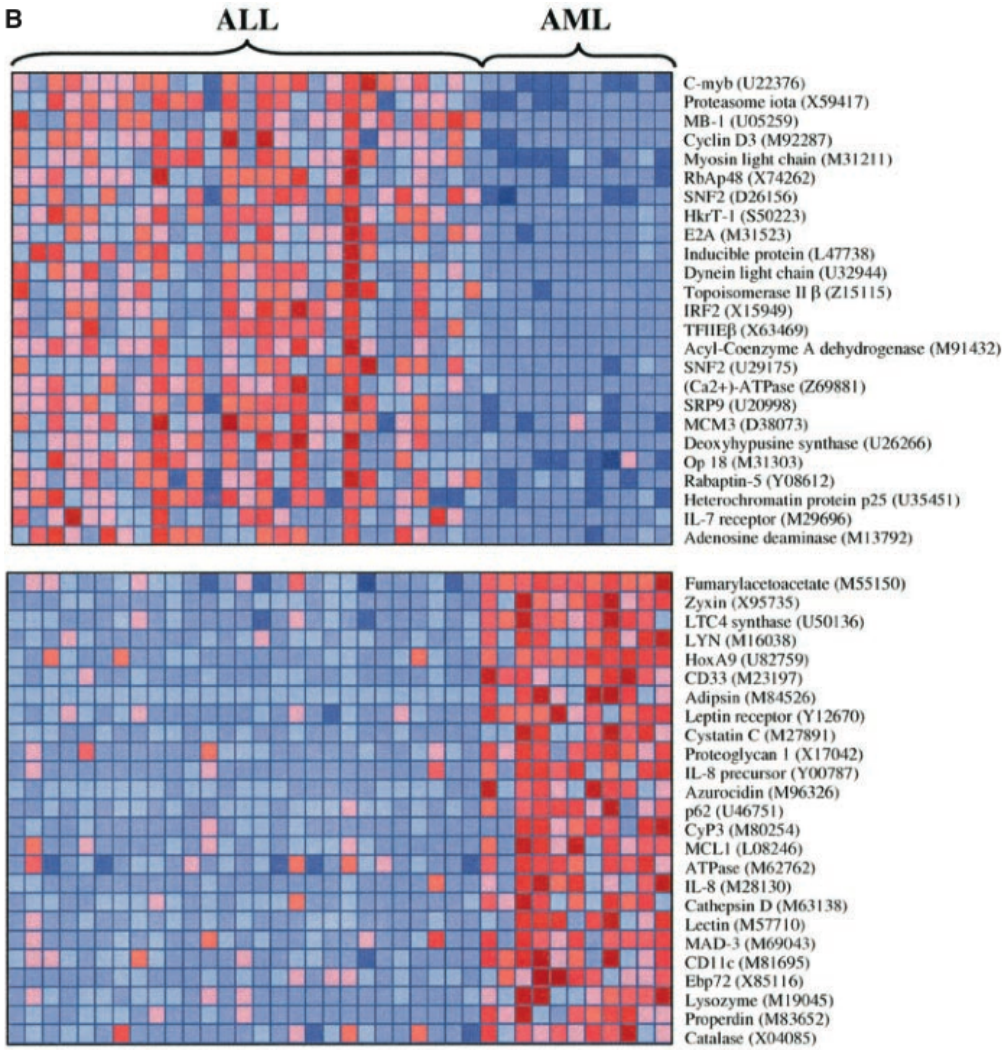
Outline

Outline

Outline

Outline

Motivation



Goal

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

Difficulty

- Large dimension
- Few samples

Linear classifiers

The model

- Each sample is represented by a vector $x = (x_1, \ldots, x_p)$
- Goal: estimate a linear function:

$$f_\beta(x) = \sum_{i=1}^{p} \beta_i x_i + \beta_0 \ .$$

- Interpretability: the weight β_i quantifies the influence of feature i (but...)

Training the model

$$f_\beta(x) = \sum_{i=1}^{p} \beta_i x_i + \beta_0 \ .$$

- Minimize an empirical risk on the training samples:

$$\min_{\beta \in \mathbb{R}^{p+1}} R_{emp}(\beta) = \frac{1}{n} \sum_{i=1}^{n} l(f_\beta(x_i), y_i) \ ,$$

- ... subject to some constraint on β, e.g.:

$$\Omega(\beta) \leq C \ .$$

Example : Norm Constraints

The approach

A common method in statistics to learn with few samples in high dimension is to constrain the Euclidean norm of β

$$\Omega_{ridge}(\beta) = \| \beta \|_2^2 = \sum_{i=1}^{p} \beta_i^2 \, ,$$

(ridge regression, support vector machines...)

Pros

- Good performance in classification

Cons

- Limited interpretation (small weights)
- No prior biological knowledge

Example : Feature Selection

The approach

Constrain most weights to be 0, i.e., select a few genes (< 100) whose expression are sufficient for classification.

- Greedy feature selection (T-tests, ...)
- Contrain the norm of β: LASSO penalty ($\| \beta \|_1 = \sum_{i=1}^{p} | \beta_i |$), elastic net penalty ($\| \beta \|_1 + \| \beta \|_2$), ...)

Pros

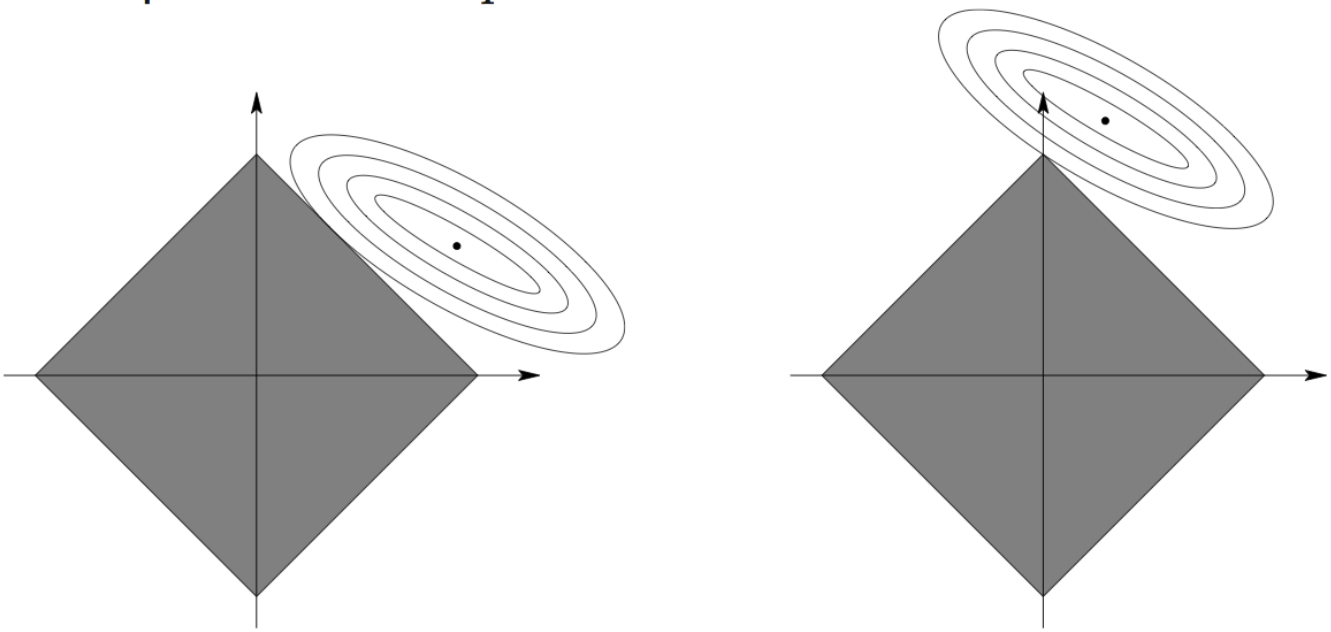
- Good performance in classification
- Biomarker selection
- Interpretability

Cons

- The gene selection process is usually not robust
- No use of prior biological knowledge

Why LASSO leads to sparse solutions

Geometric interpretation with $p = 2$



Incorporating prior knowledge

The idea

- If we have a specific prior knowledge about the "correct" weights, it can be included in Ω in the contraint:

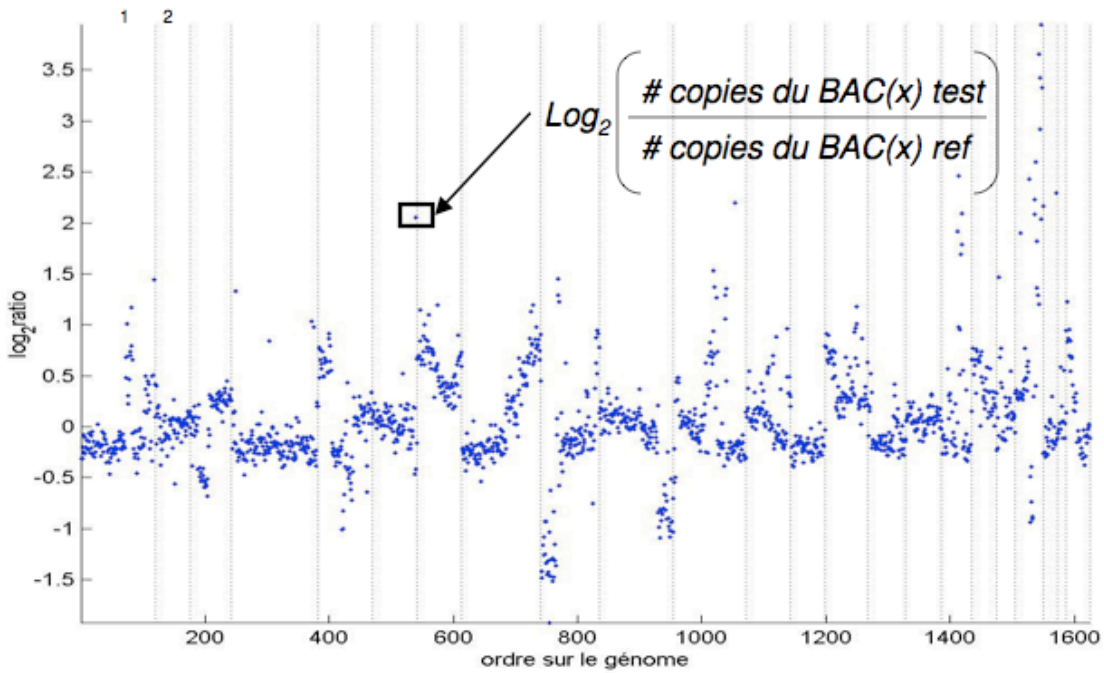
$$\text{Minimize } R_{emp}(\beta) \text{ subject to } \Omega(\beta) \leq C\,.$$

- If we design a convex function Ω, then the algorithm boils down to a convex optimization problem (usually easy to solve).
- Similar to priors in Bayesian statistics

Example: CGH array classification

Motivation

- Comparative genomic hybridization (CGH) data measure the DNA copy number along the genome
- Very useful, in particular in cancer research
- Can we classify CGH arrays for diagnosis or prognosis purpose?



Jain et al. Genome research 2002 12:325-332

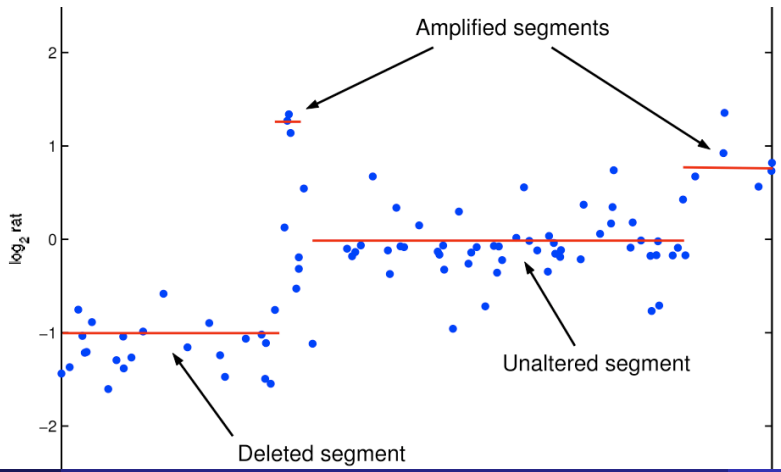
Example: CGH array classification

Prior knowledge

- Let **x** be a CGH profile
- We focus on linear classifiers, i.e., the sign of :

$$f(\mathbf{x}) = \mathbf{x}^\top \beta \,.$$

- We expect β to be
  - sparse : only a few positions should be discriminative
  - piecewise constant : within a region, all probes should contribute equally

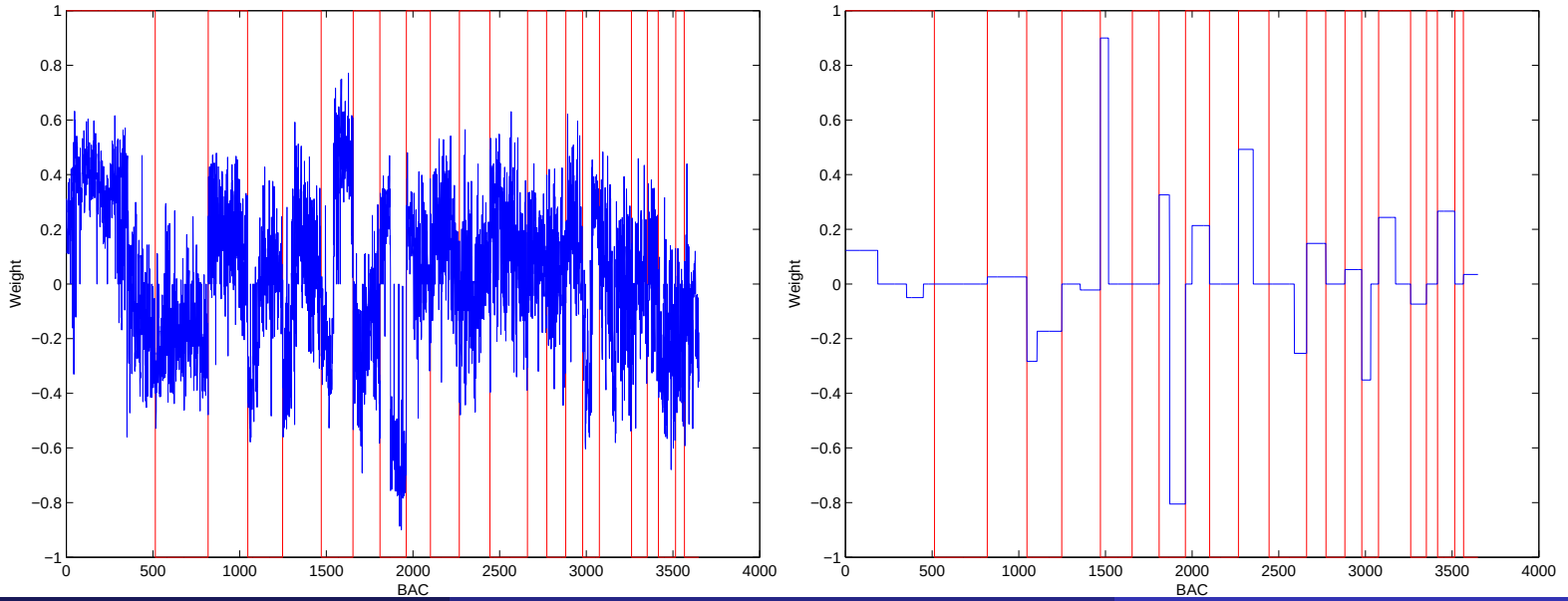


Example: CGH array classification

A solution (Rapaport et al., 2008)

$$\Omega_{fusedlasso}(\beta) = \sum_i |\beta_i| + \sum_{i \sim j} |\beta_i - \beta_j| \,.$$

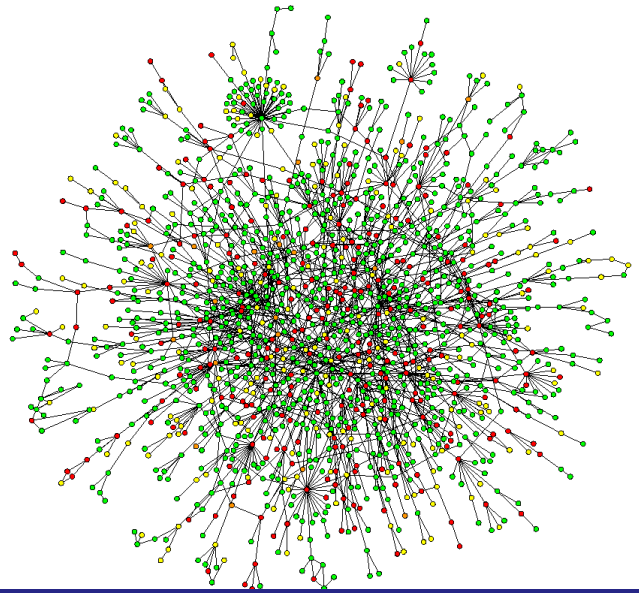
- Good performance on diagnosis for bladder cancer, and prognosis for melanoma.
- More interpretable classifiers



Example: finding discriminant modules in gene networks

The problem

- Classification of gene expression: too many genes
- A gene network is given (PPI, metabolic, regulatory, signaling, co-expression...)
- We expect that “clusters of genes” (modules) in the network contribute similarly to the classification



Example: finding discriminant modules in gene networks

Prior hypothesis

Genes near each other on the graph should have similar weigths.

Two solutions (Rapaport et al., 2007, 2008)

$$\Omega_{spectral}(\beta) = \sum_{i \sim j} (\beta_i - \beta_j)^2 \,,$$

$$\Omega_{graphfusion}(\beta) = \sum_{i \sim j} |\beta_i - \beta_j| + \sum_i |\beta_i| \,.$$

Example: finding discriminant modules in gene networks

Prior hypothesis

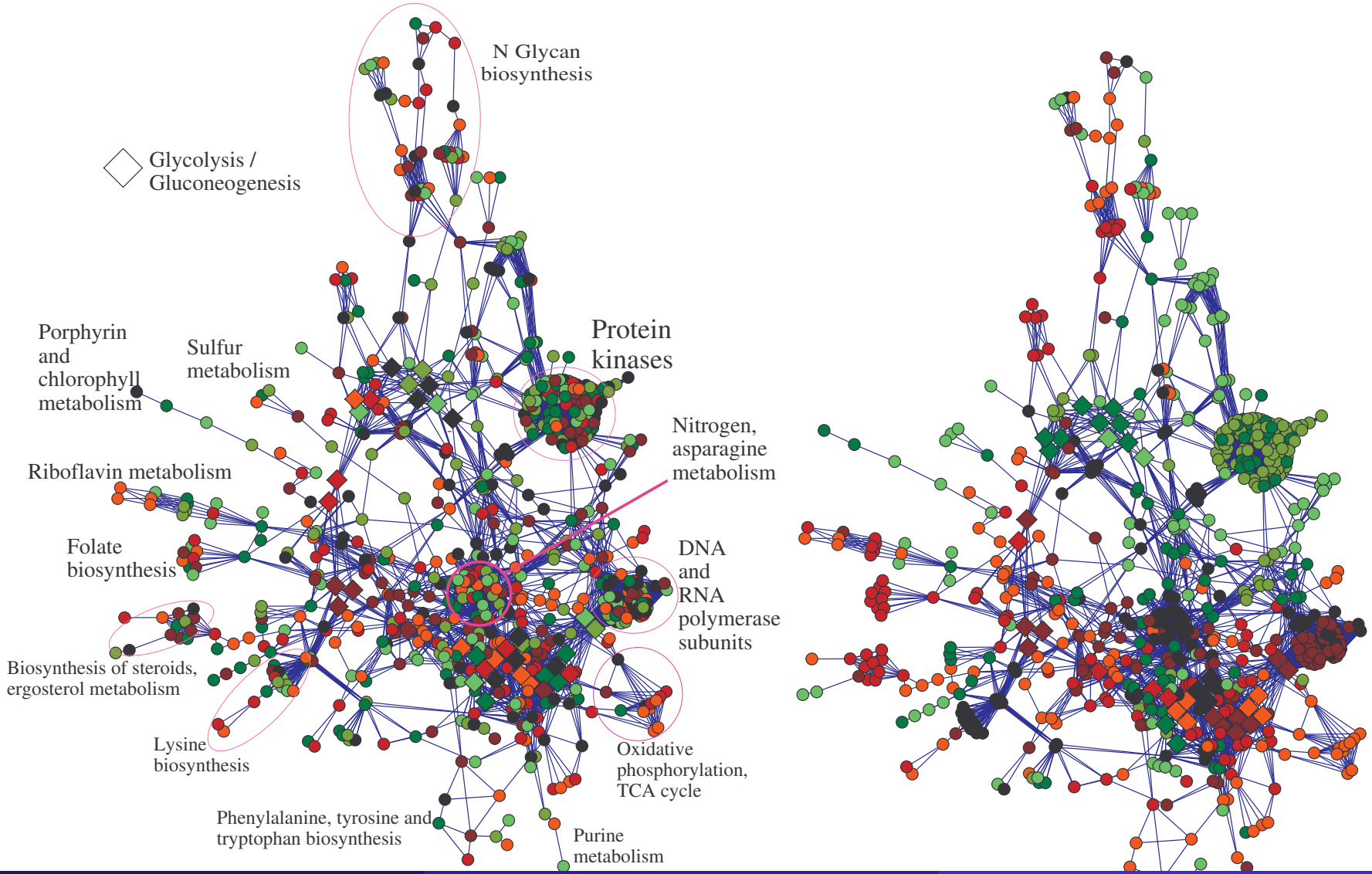
Genes near each other on the graph should have similar weigths.

Two solutions (Rapaport et al., 2007, 2008)

$$\Omega_{spectral}(\beta) = \sum_{i \sim j} (\beta_i - \beta_j)^2 \,,$$

$$\Omega_{graphfusion}(\beta) = \sum_{i \sim j} |\beta_i - \beta_j| + \sum_i |\beta_i| \,.$$

Example: finding discriminant modules in gene networks



Example: finding discriminant modules in gene networks

Prior hypothesis

Genes near each other on the graph should have non-zero weigths (i.e., the support of β should be made of a few connected components).

Two solutions?

$$\Omega_{intersection}(\beta) = \sum_{i \sim j} \sqrt{\beta_i^2 + \beta_j^2}\,,$$

$$\Omega_{union}(\beta) = \sup_{\alpha \in \mathbb{R}^p : \forall i \sim j, \|\alpha_i^2 + \alpha_j^2\| \leq 1} \alpha^\top \beta\,.$$

Example: finding discriminant modules in gene networks

Prior hypothesis

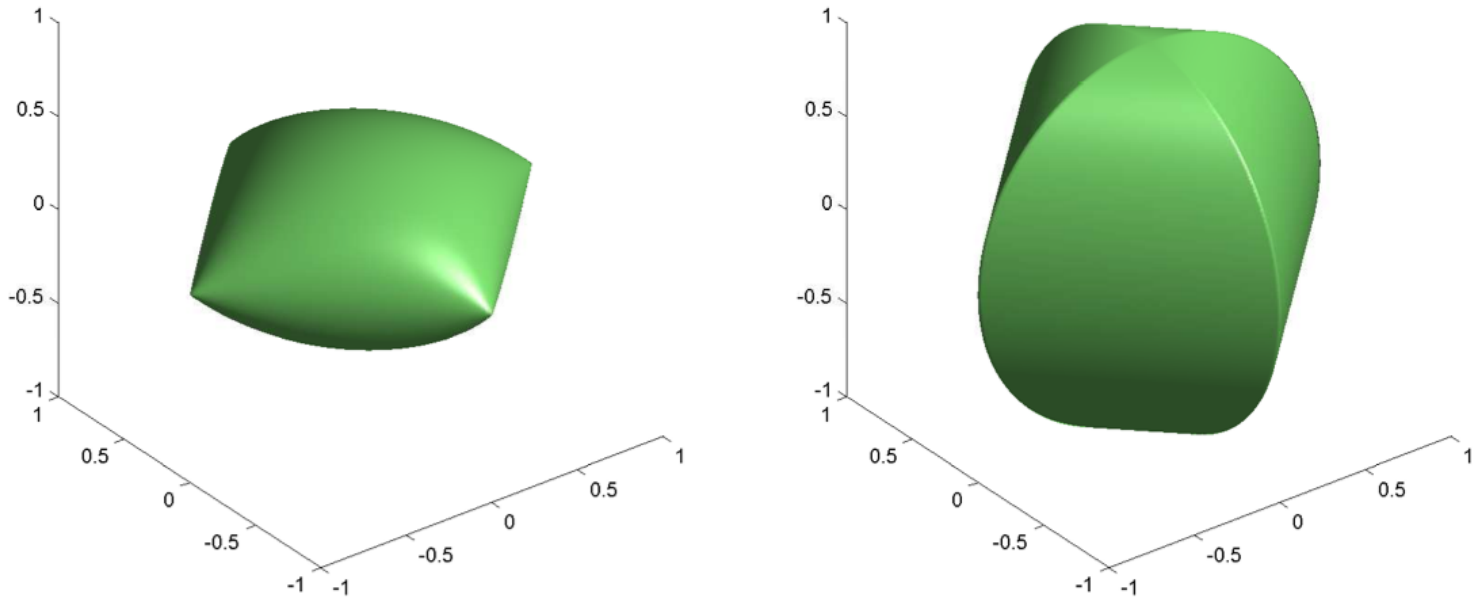
Genes near each other on the graph should have non-zero weigths (i.e., the support of β should be made of a few connected components).

Two solutions?

$$\Omega_{intersection}(\beta) = \sum_{i \sim j} \sqrt{\beta_i^2 + \beta_j^2}\,,$$

$$\Omega_{union}(\beta) = \sup_{\alpha \in \mathbb{R}^p : \forall i \sim j, \|\alpha_i^2 + \alpha_j^2\| \leq 1} \alpha^\top \beta\,.$$

Example: finding discriminant modules in gene networks



Groups $(1,2)$ *and* $(2,3)$. *Left:* $\Omega_{intersection}(\beta)$. *Right:* $\Omega_{union}(\beta)$. *Vertical axis is* β_2.

Outline

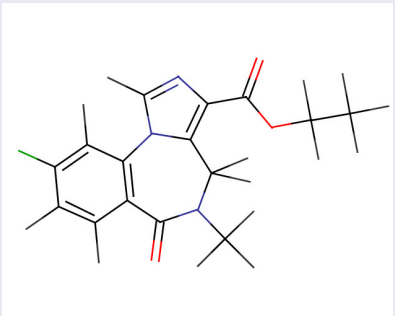
Ligand-Based Virtual Screening

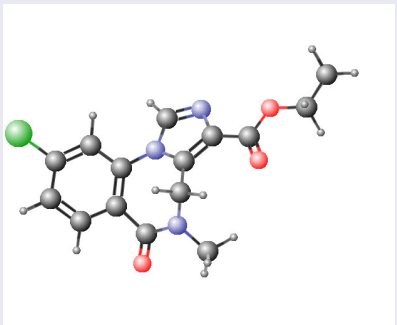
Objective

Build models to predict biochemical properties of small molecules from their structures.

Structures

$C_{15}H_{14}ClN_3O_3$

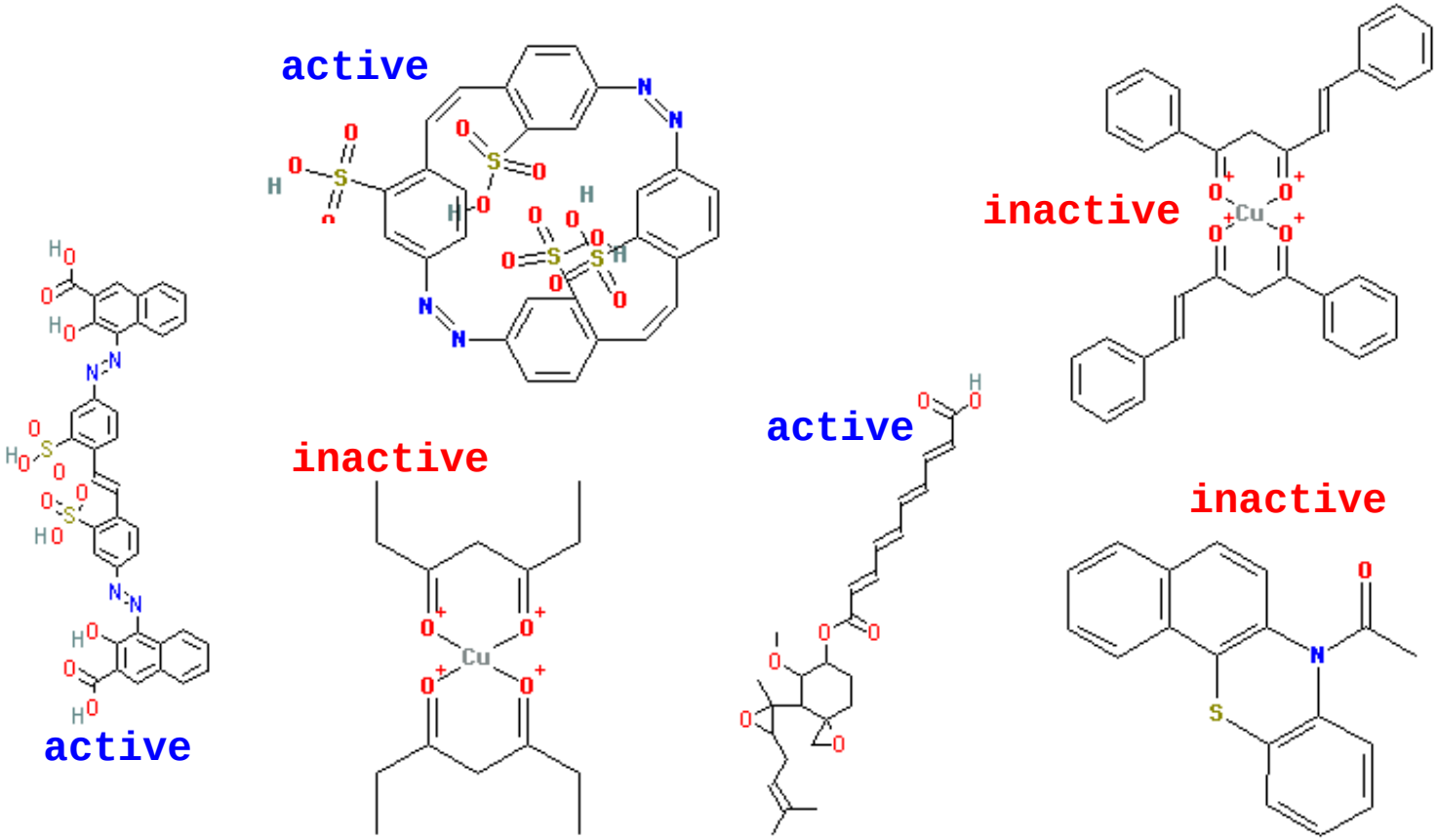




Properties

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

Ligand-Based Virtual Screening and QSAR



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

Formalization

The problem

- Given a set of training instances $(x_1, y_1), \ldots, (x_n, y_n)$, where x_i's are graphs and y_i's are continuous or discrete variables of interest,
- Estimate a function

$$y = f(x)$$

where x is any graph to be labeled.

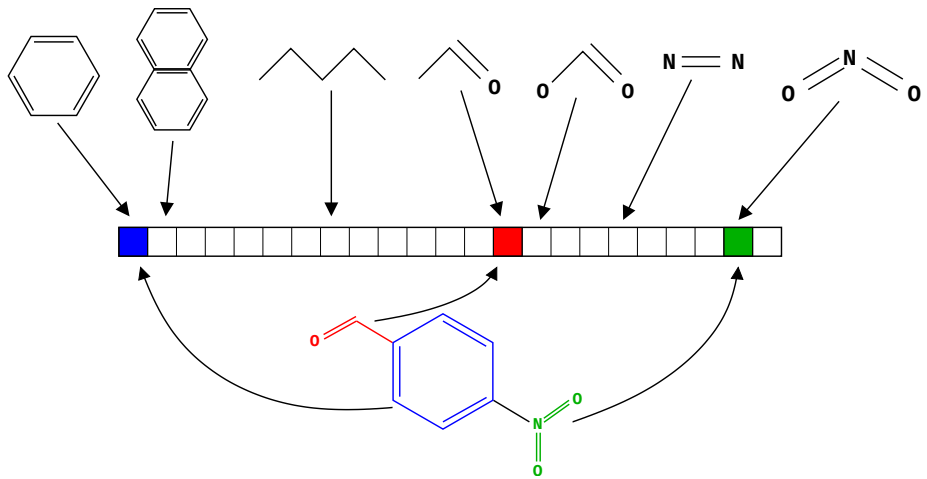
- This is a classical regression or pattern recognition problem over the set of graphs.

Classical approaches

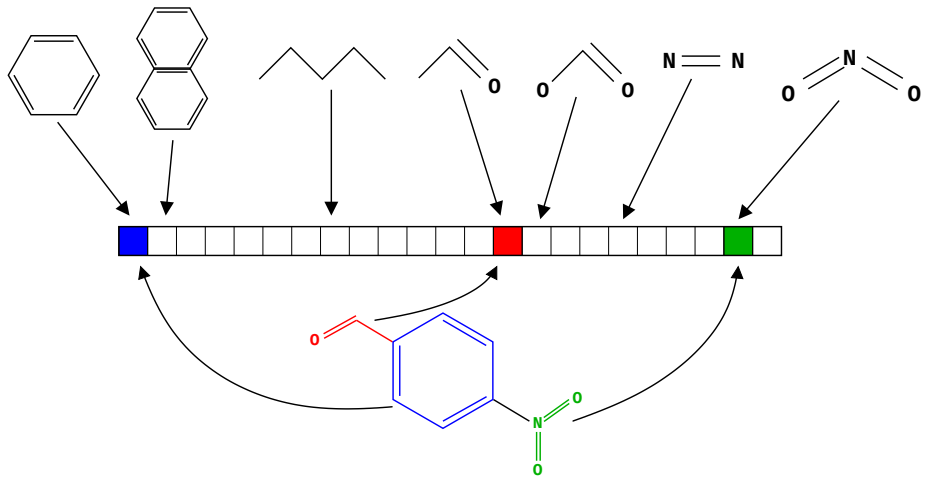
Two steps

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

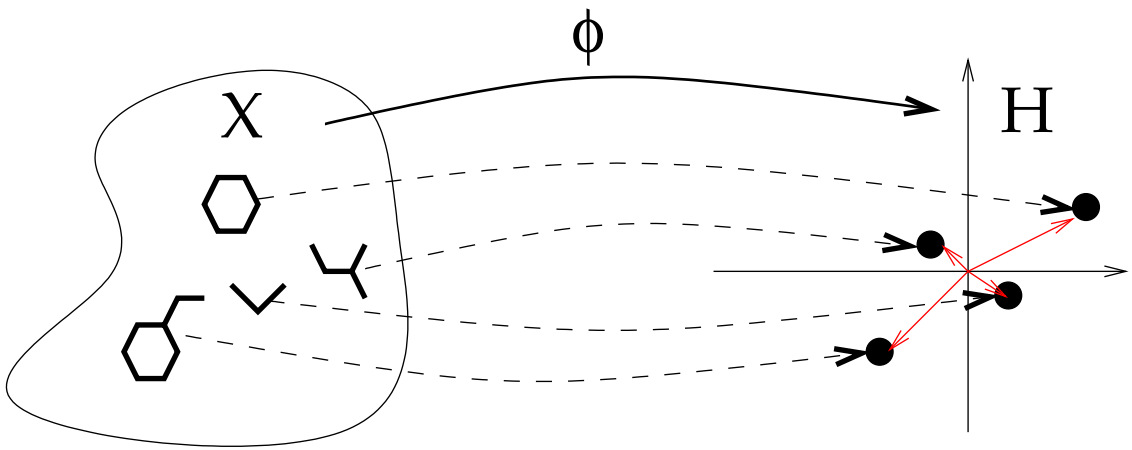
- Many descriptors are needed to characterize various features (in particular for 2D and 3D descriptors)
- But too many descriptors are harmful for memory storage, computation speed, statistical estimation

Kernels

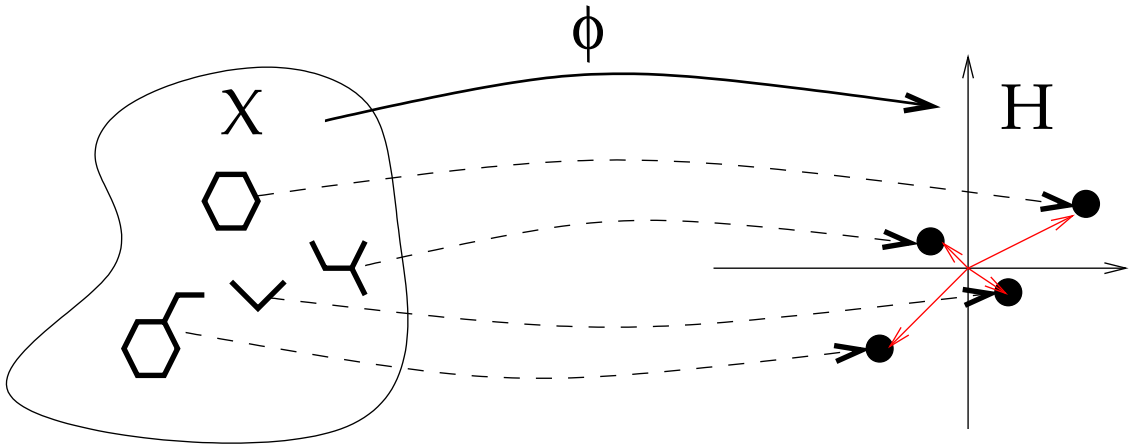
Definition

- 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



$$K(x, x') = \Phi(x)^\top \Phi(x')$$

The trick

- Many linear algorithms for regression or pattern recognition can be expressed only in terms of inner products between vectors
- Computing the kernel is often more efficient than computing $\Phi(x)$, especially in high or infinite dimensions!

Expressiveness vs Complexity of graph kernels

Definition: Complete graph kernels

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

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

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

Proposition (Gärtner et al., 2003)

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

Expressiveness vs Complexity of graph kernels

Definition: Complete graph kernels

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

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

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

Proposition (Gärtner et al., 2003)

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

Subgraph kernel

Definition

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

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

- The subgraph kernel between any two graphs G_1 and G_2 is defined by:

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

Subgraph kernel complexity

Proposition (Gärtner et al., 2003)

Computing the subgraph kernel is NP-hard when:

- $\mathcal{X}$ is the set of all graphs (all subgraph kernel)
- $\mathcal{X}$ is the set of all linear graphs (path kernel)

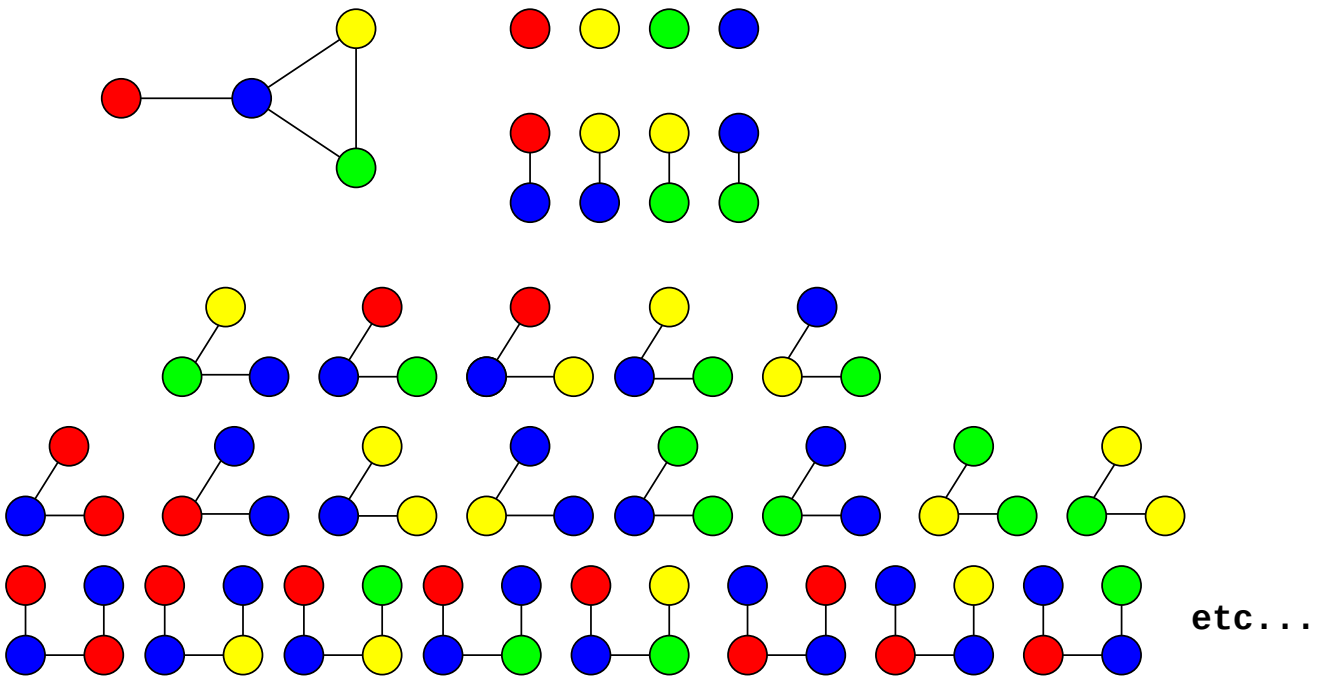
Proof (sketch)

Computing these kernels allows to decide whether a graph has a Hamiltonian path, which a NP-complete.

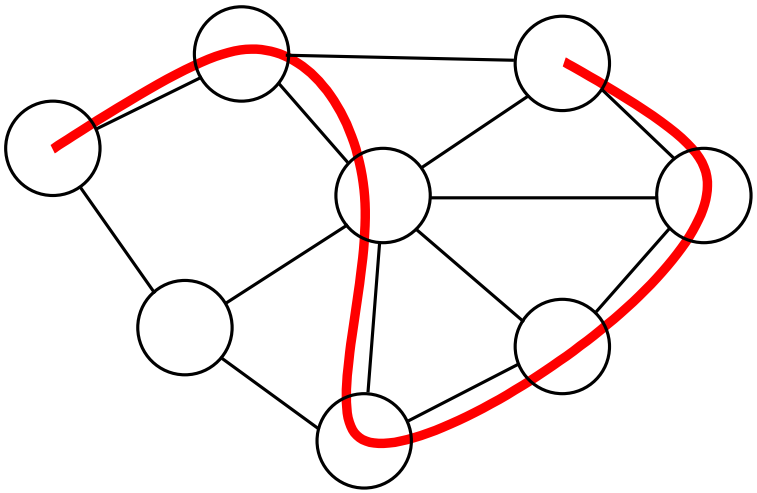
Walks

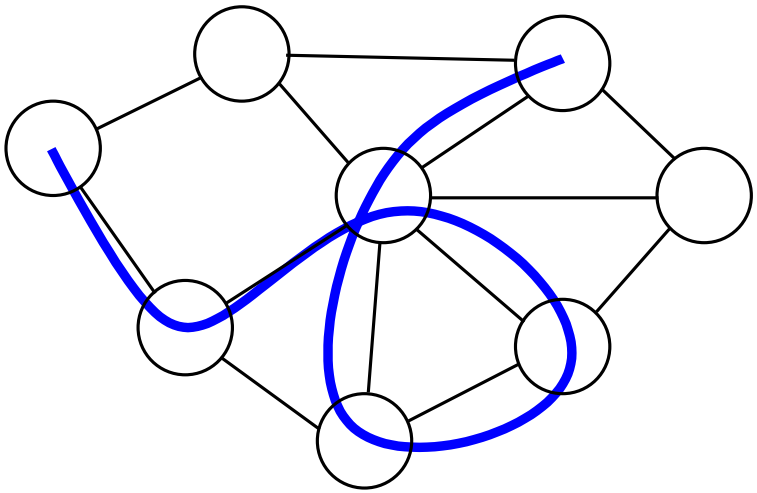
Definition

- 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 $\mathcal{X}$ 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 $\mathcal{X}$ 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 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 (Kashima et al., 2003).

- 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 (Gärtner et al., 2003).

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 (Kashima et al., 2003).

- 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 (Gärtner et al., 2003).

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 (Kashima et al., 2003).

- 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 (Gärtner et al., 2003).

Computation of walk kernels

Proposition

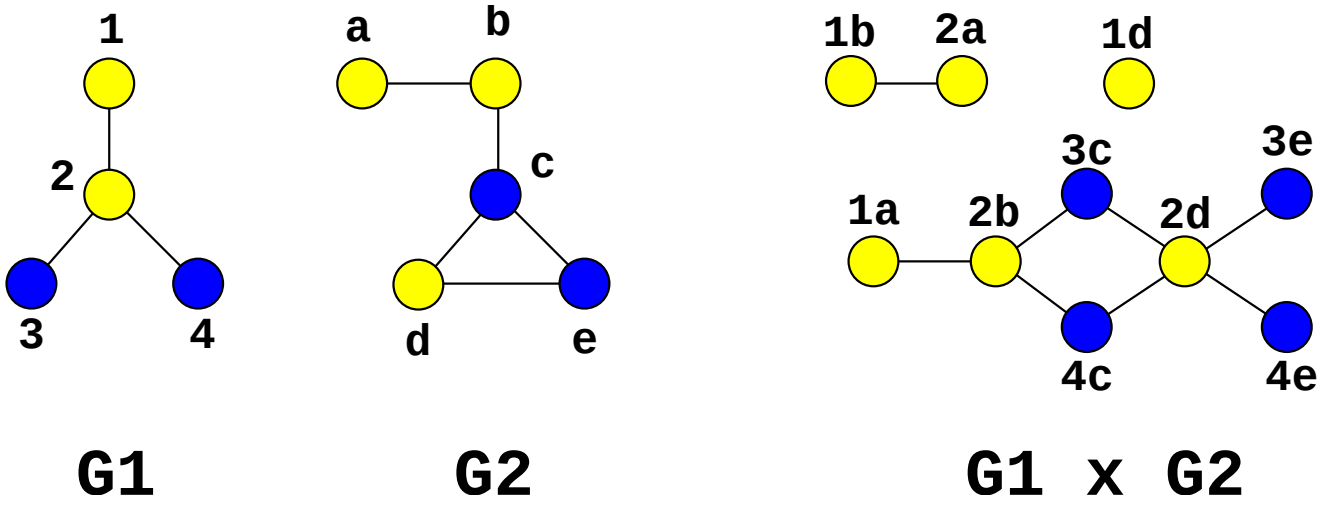
These three kernels (*n*th-order, random and geometric walk kernels) can be computed efficiently in polynomial time.

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

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

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:

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

- 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:

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

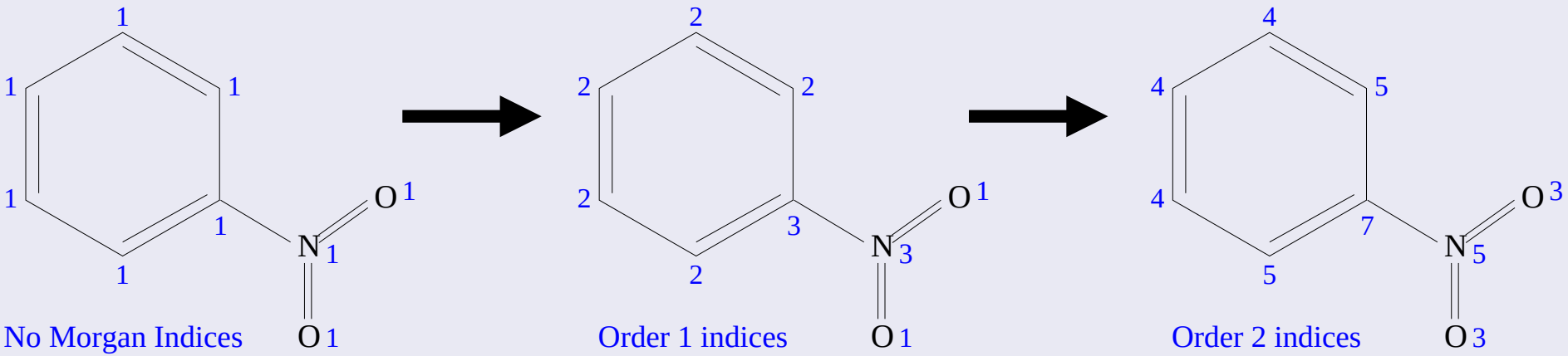
- Let Λ_i be the vector of $\lambda^i(v)$ and Λ_t be the matrix of $\lambda^t(v, v')$:

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

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

Extensions 1: label enrichment

Atom relabebling with the Morgan index

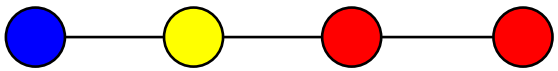


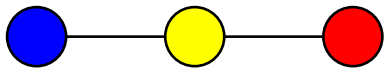
- Compromise between fingerprints and structural keys features.
- Other relabeling schemes are possible (graph coloring).
- Faster computation with more labels (less matches implies a smaller product graph).

Extension 2: Non-tottering walk kernel

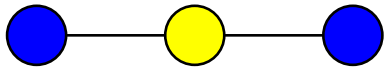
Tottering walks

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





Non-tottering

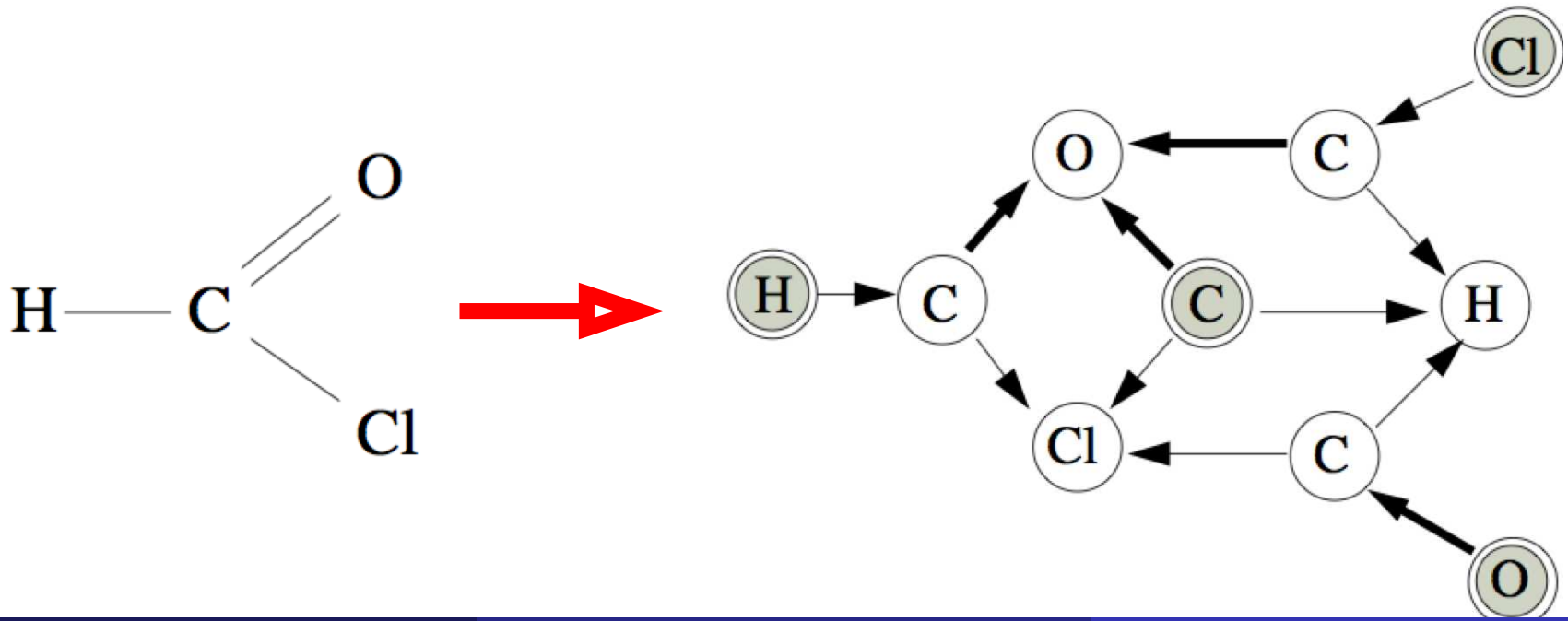


Tottering

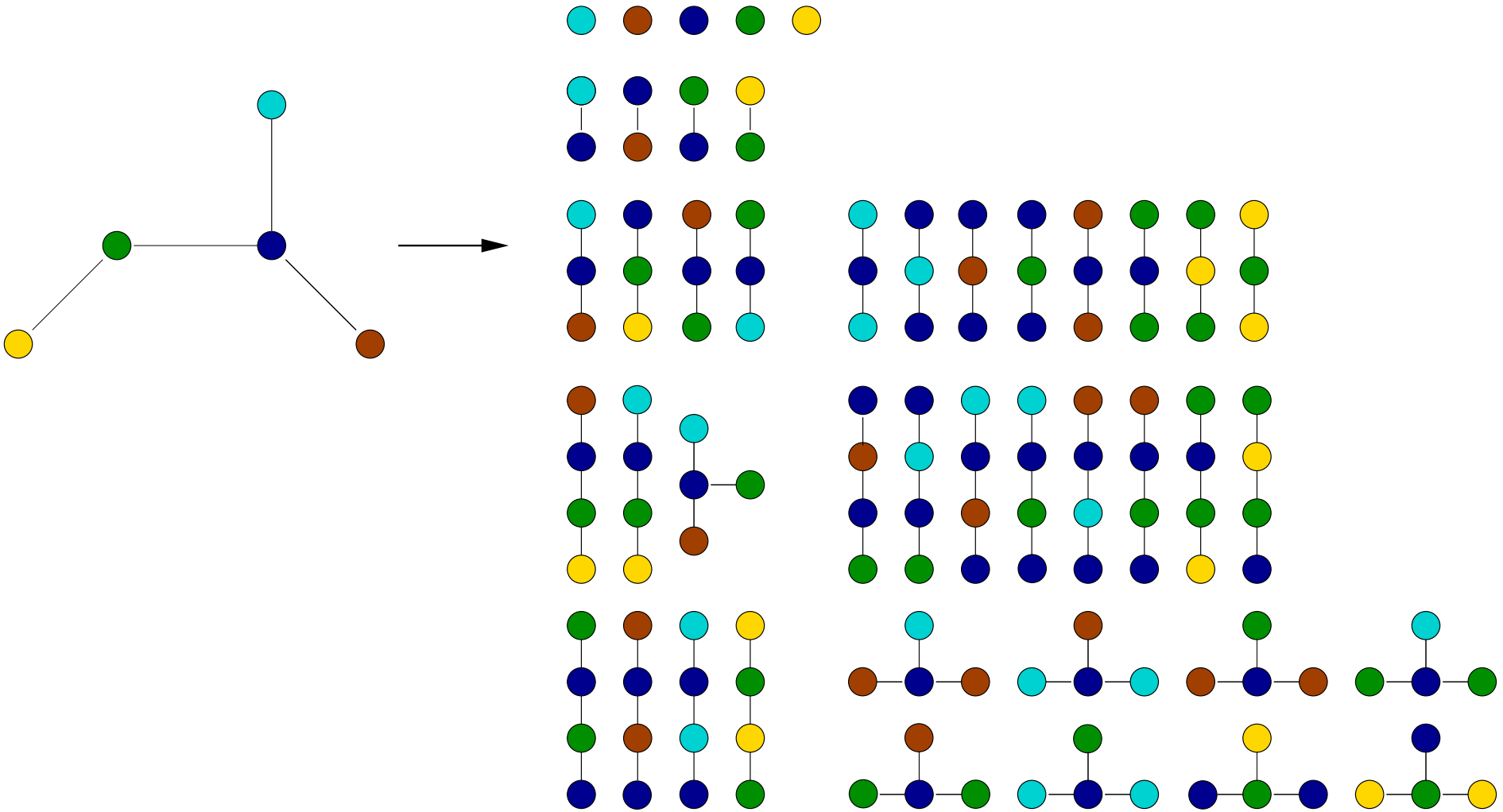
- Tottering walks seem irrelevant for many applications
- Focusing on non-tottering walks is a way to get closer to the path kernel (e.g., equivalent on trees).

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

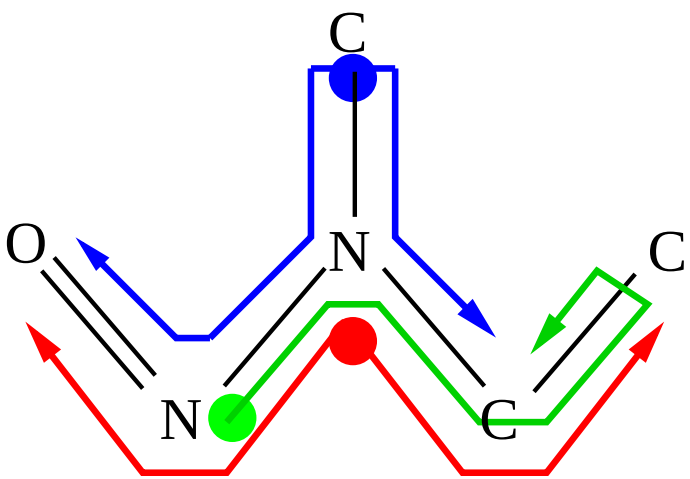
- Second-order Markov random walk to prevent tottering walks
- Written as a first-order Markov random walk on an augmented graph
- Normal walk kernel on the augmented graph (which is always a directed graph).

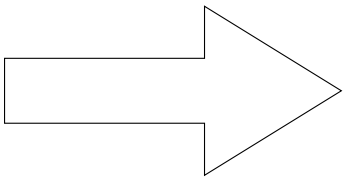


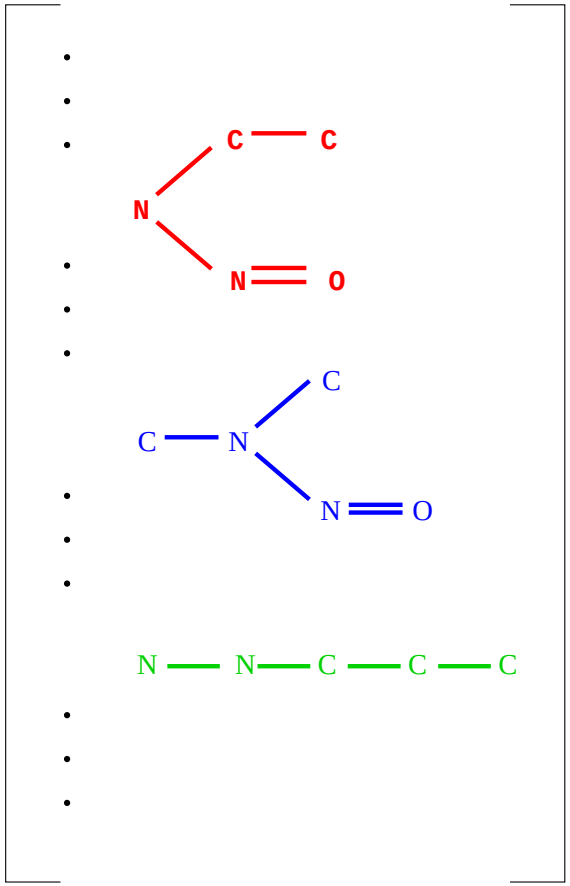
Extension 2: Subtree kernels



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.

Application (Mahé et al., 2004)

MUTAG dataset

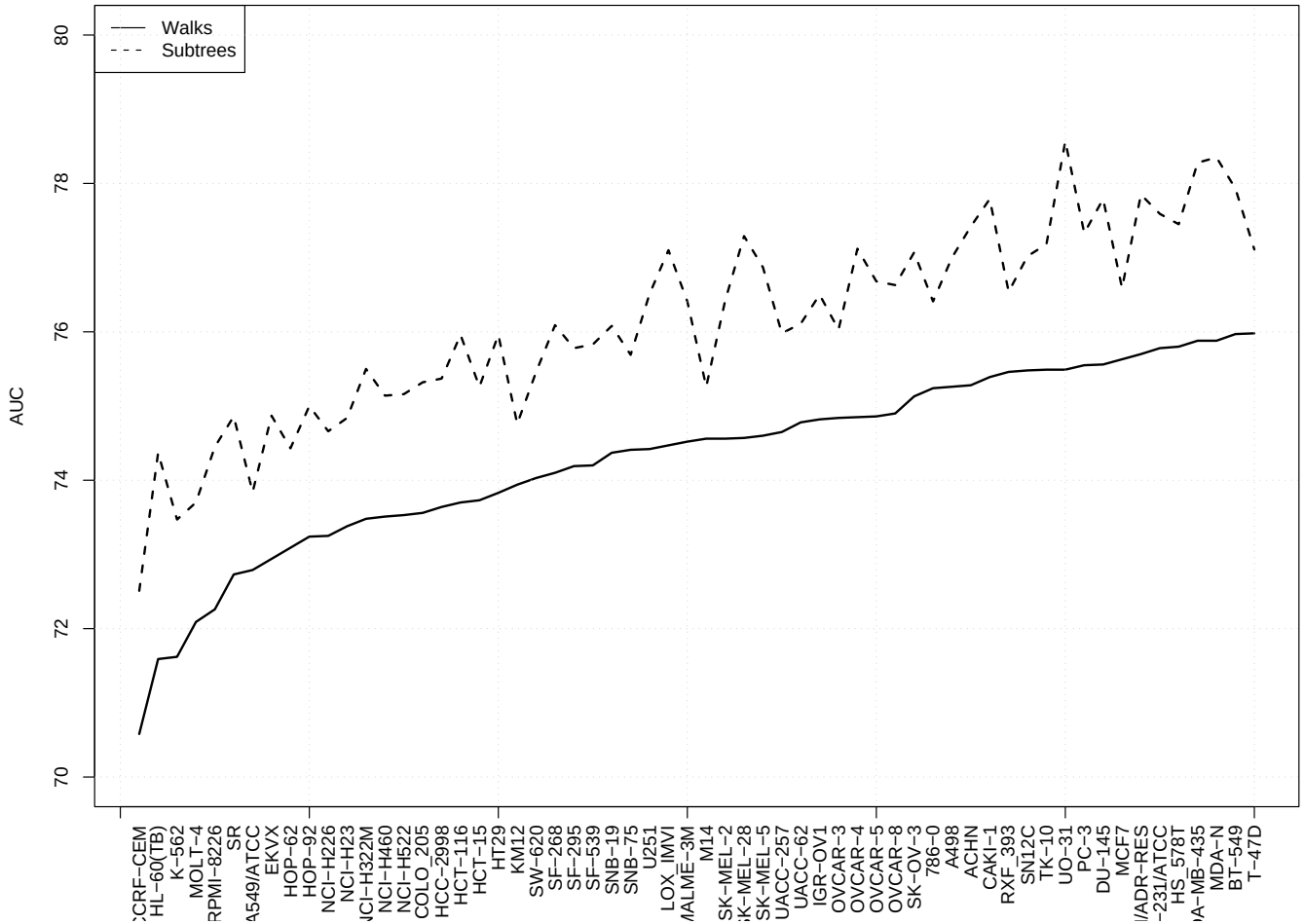
- aromatic/hetero-aromatic compounds
- high mutagenic activity /no mutagenic activity, assayed in *Salmonella typhimurium*.
- 188 compouunds: 125 + / 63 -

Results

10-fold cross-validation accuracy

Method	Accuracy
Progol1	81.4%
2D kernel	91.2%

2D Subtree vs fragment kernels (Mahé and V, 2007)



Screening of inhibitors for 60 cancer cell lines (from Mahé and V., 2008)

Outline

Conclusion

- Modern machine learning methods for regression / classification lend themselves well to the integration of prior knowledge in the penalization / regularization function.
- Kernel methods (eg SVM) allow to manipulate complex objects (eg molecules, biological sequences) as soon as kernels can be defined and computed.

People I need to thank

Including prior knowledge in penalization

Franck Rapaport, Emmanuel Barillot, Andrei Zynoviev, Laurent Jacob, Kevin Bleakley...

Virtual screening, kernels etc..

Pierre Mahé, Laurent Jacob, Liva Ralaivola, Véronique Stoven