Machine Learning in Computational and Systems Biology

Jean-Philippe Vert
Jean-Philippe.Vert@mines-paristech.fr

Mines ParisTech / Institut Curie / Inserm

Malaysia Genome Institute, Nov 8-12, 2010.

Outline

Outline

Outline

Outline

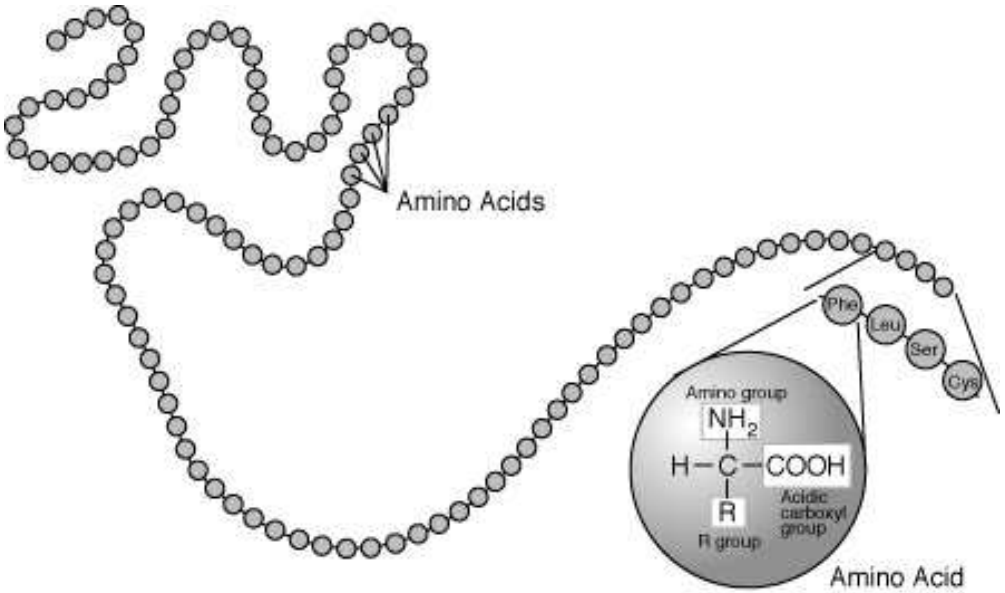
Outline

Outline

Outline

Outline

Proteins



A : Alanine	V : Valine	L : Leucine
F : Phenylalanine	P : Proline	M : Methionine
E : Acide glutamique	K : Lysine	R : Arginine
T : Threonine	C : Cysteine	N : Asparagine
H : Histidine	V : Thyrosine	W : Tryptophane
I : Isoleucine	S : Serine	Q : Glutamine
D : Acide aspartique	G : Glycine	

Protein annotation

Data available

- Secreted proteins:

```
MASKATLLLAFTLLFATCIARHQQRQQQQNQCQLQNIEA...
MARSSLFTFLCLAVFINGCLSQIEQQSPWEFQGSEVW...
MALHTVLIMLSLLPMLEAQNPEHANITIGEPITNETLGWL...
...
```

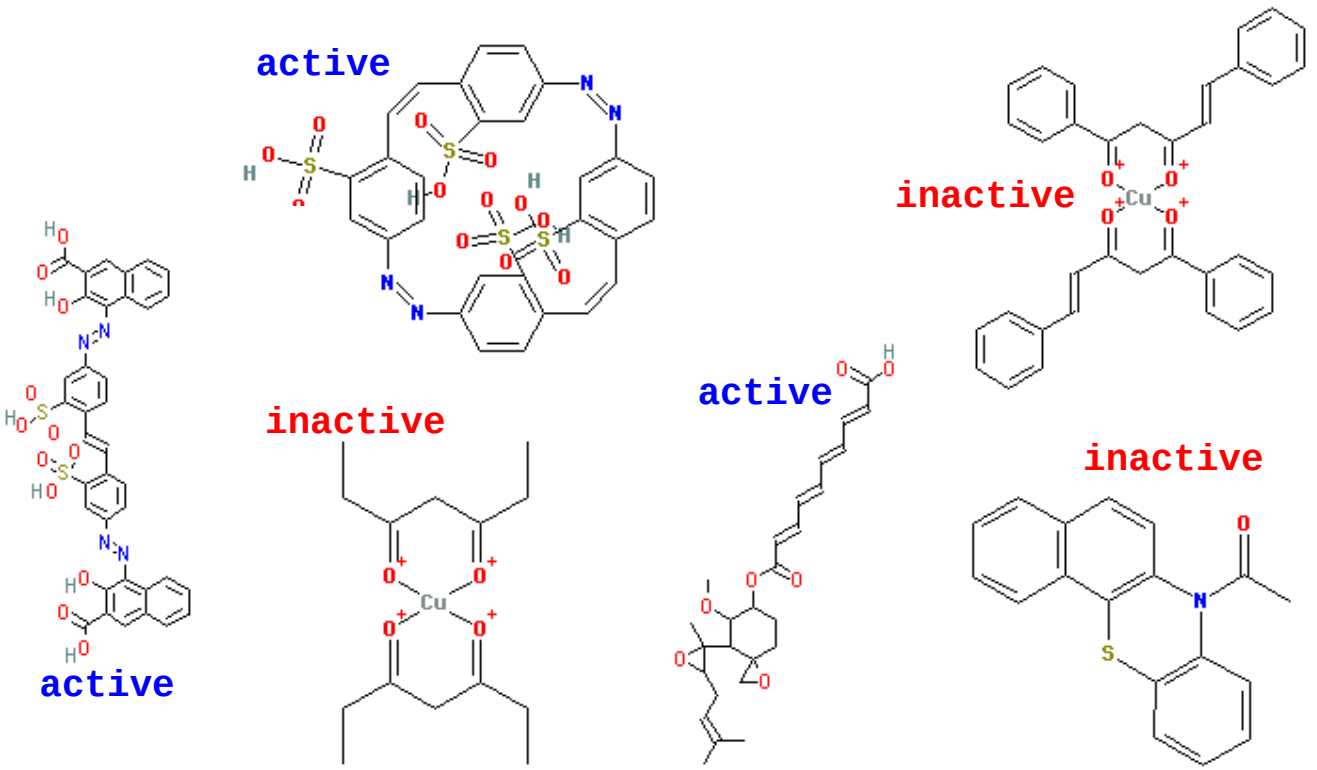
- Non-secreted proteins:

```
MAPPSVFAEVPQAQPVLVFKLIADFREDPDPRKVNLGVG...
MAHTLGLTQPNSTEPHKISFTAKEIDVIEWKGDILVVG...
MSISESYAKEIKTAFRQFTDFPIEGEQFEDFLPIIGNP..
...
```

Problem 1

Given a newly sequenced protein, is it secreted or not?

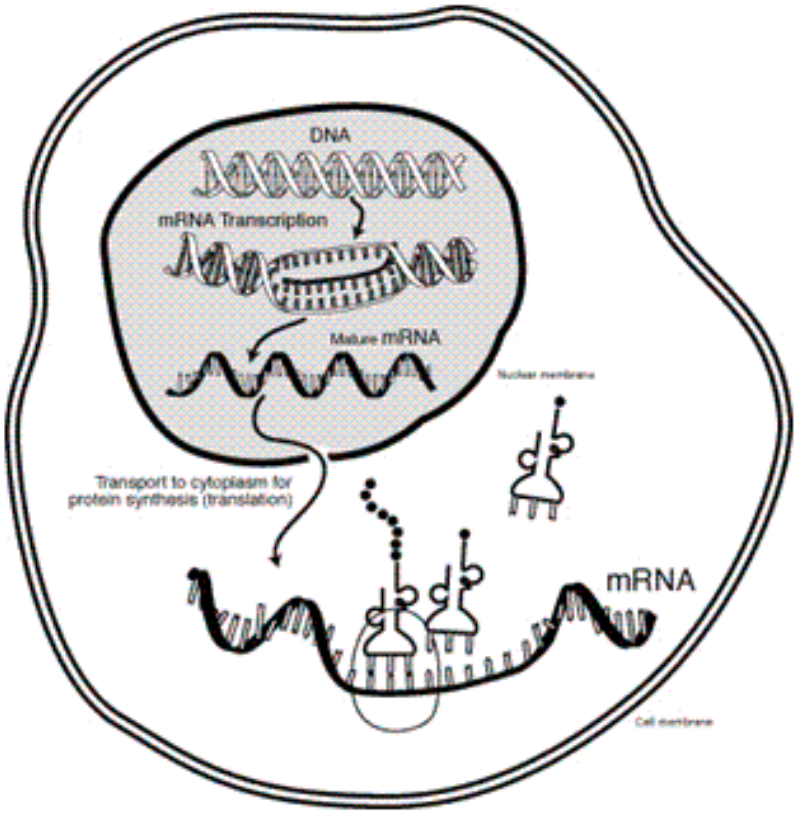
Drug discovery

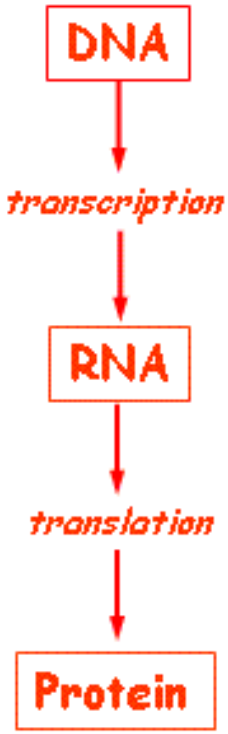


Problem 2

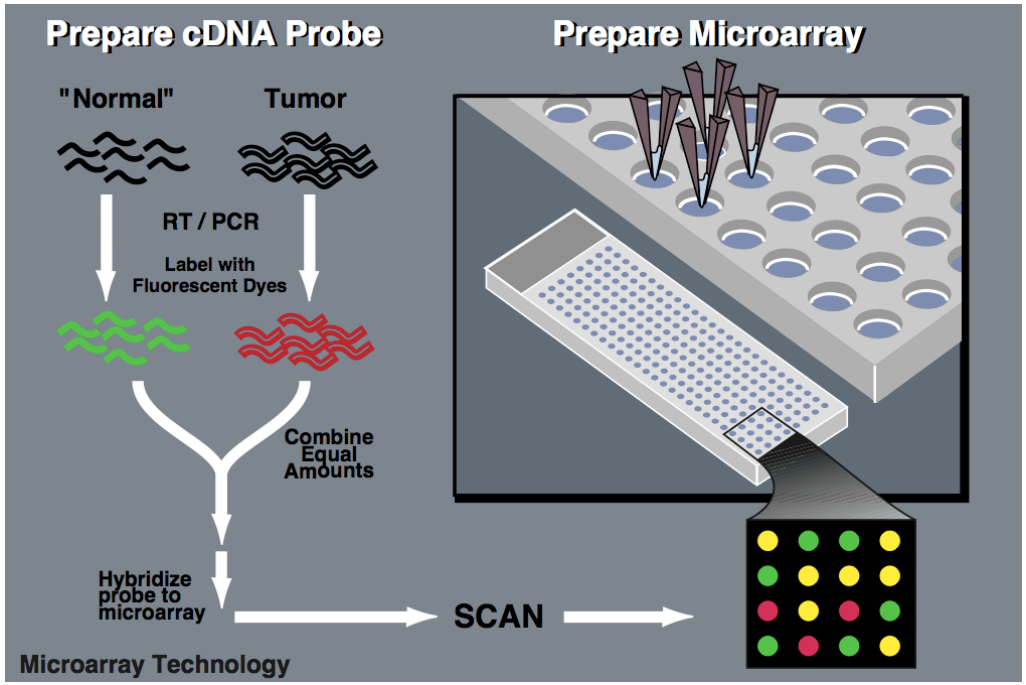
Given a new candidate molecule, is it likely to be active?

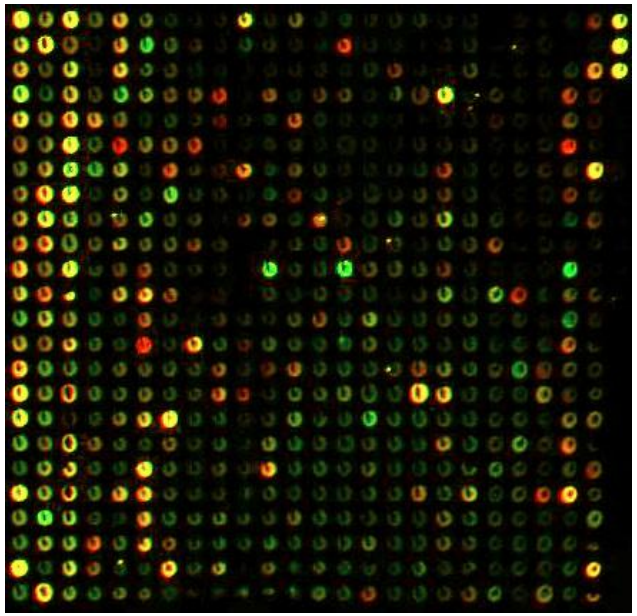
DNA → RNA → protein



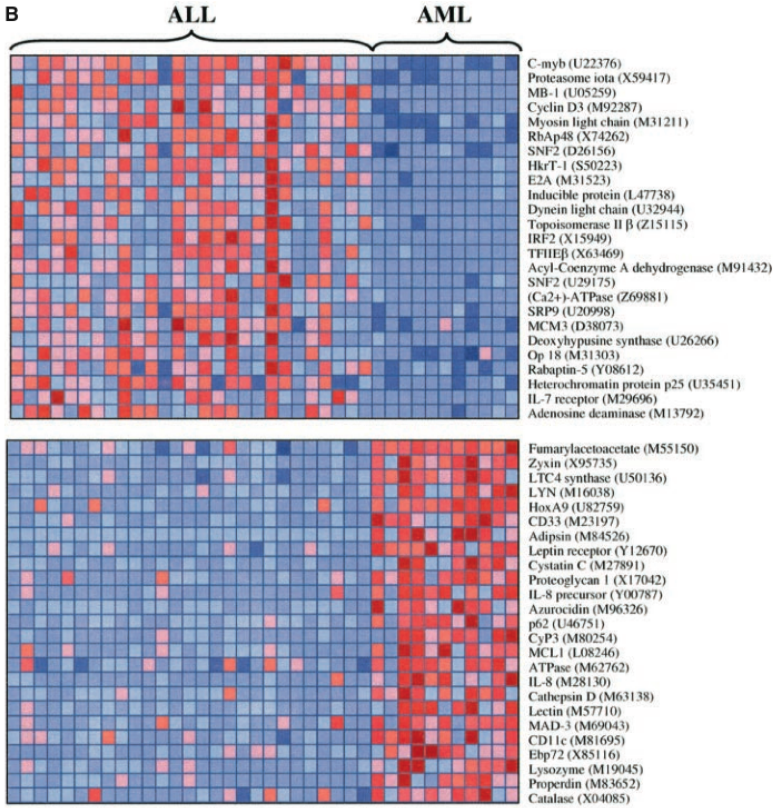


Tissue profiling with DNA chips





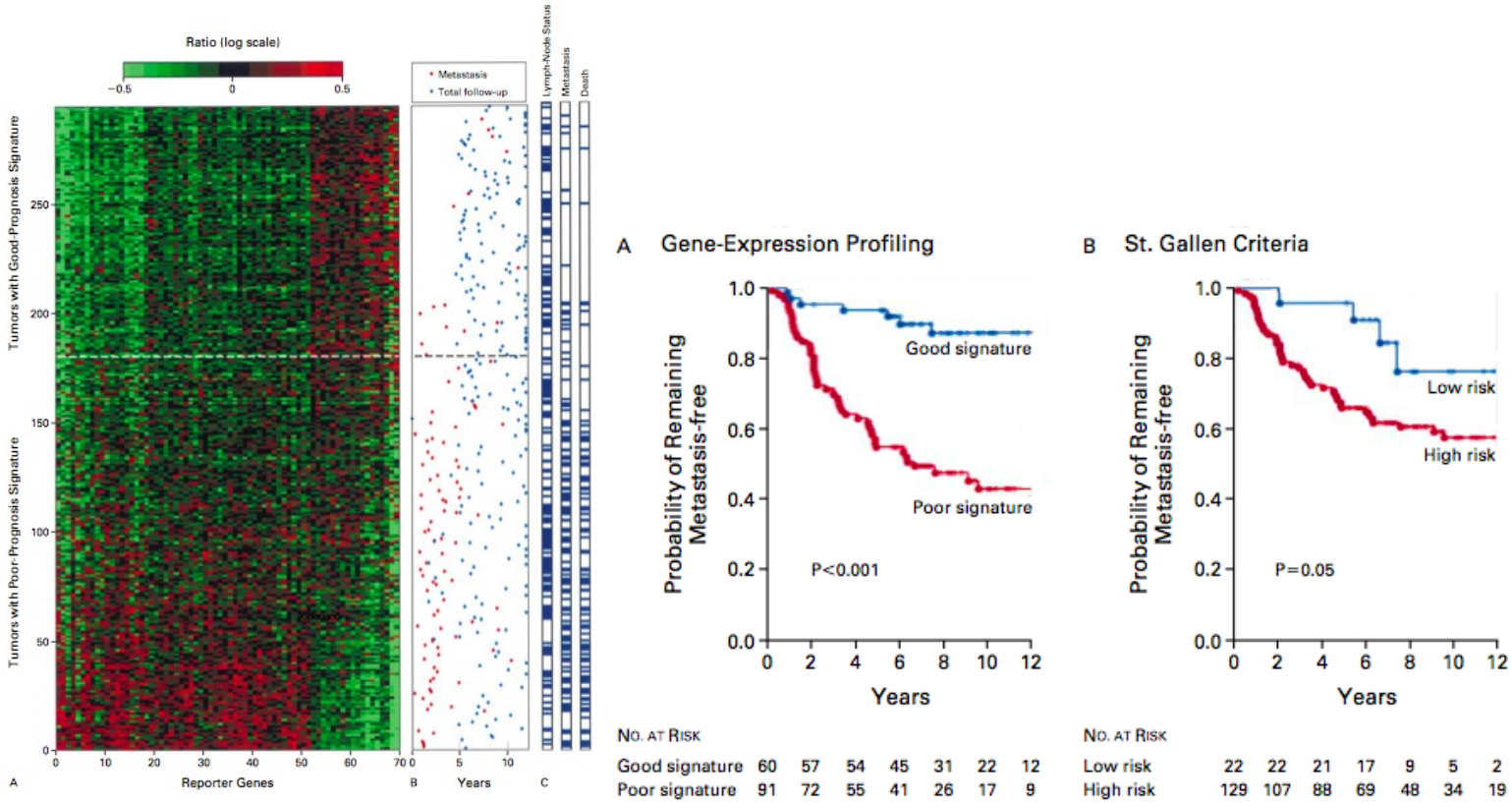
Use in diagnosis



Problem 3

Given the expression profile of a leukemia, is it an acute lymphocytic or myeloid leukemia (ALL or AML)?

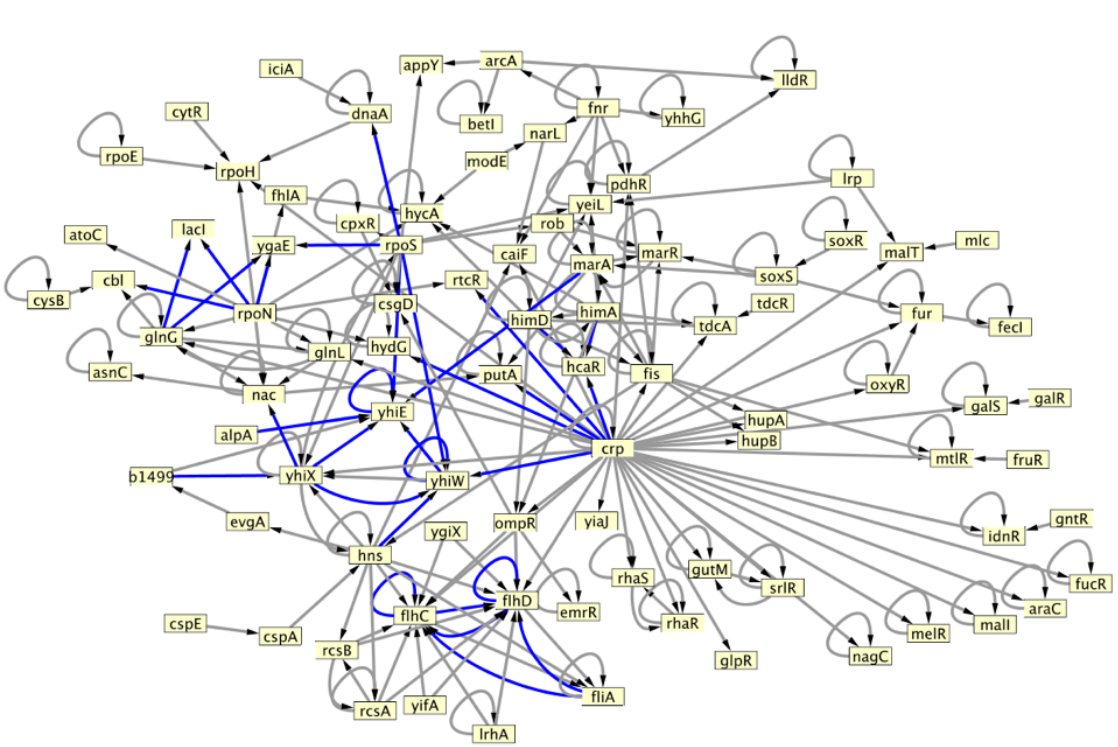
Use in prognosis

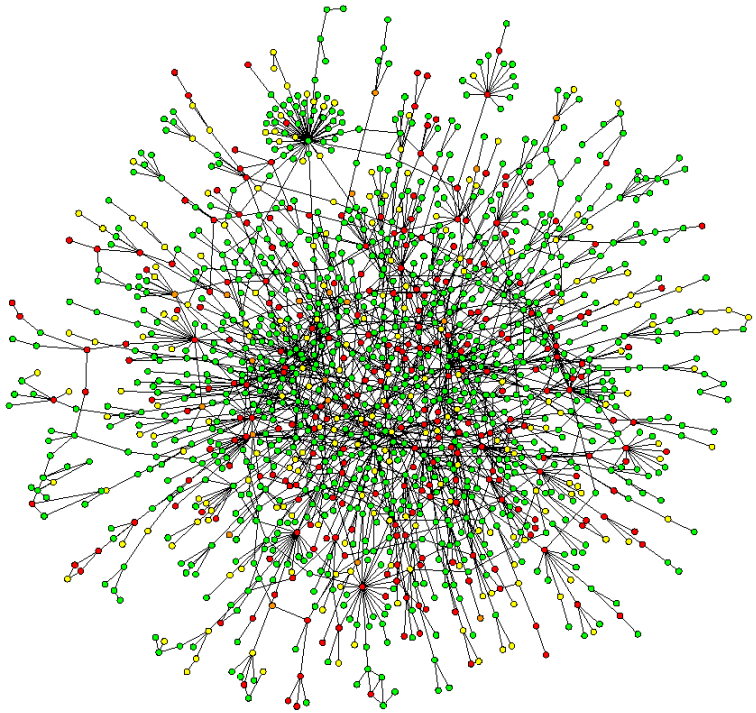


Problem 4

Given the expression profile of a breast cancer, is the risk of relapse within 5 years high?

Gene network inference

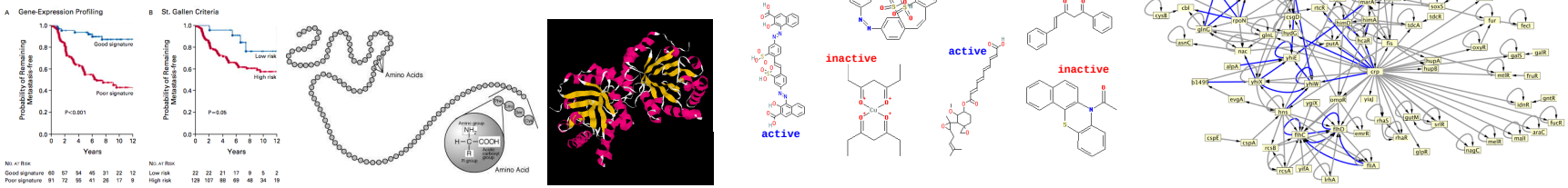


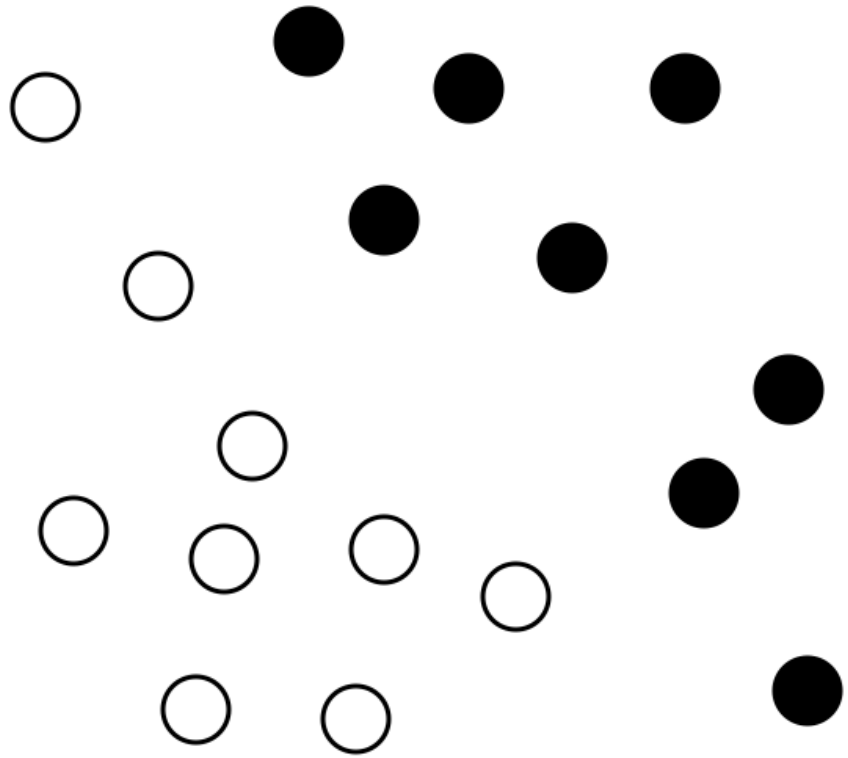


Problem 5

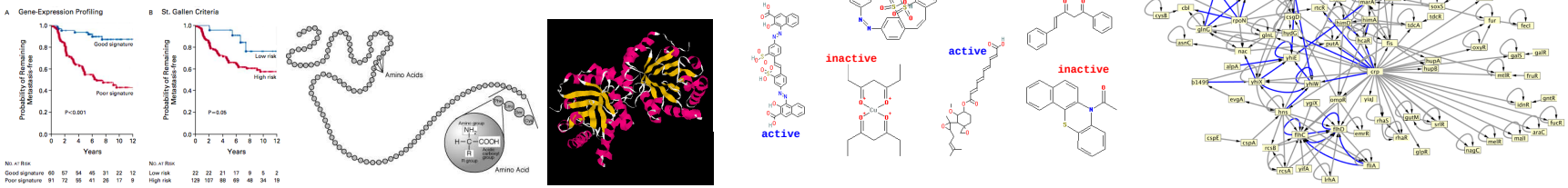
Given known interactions, can we infer new ones?

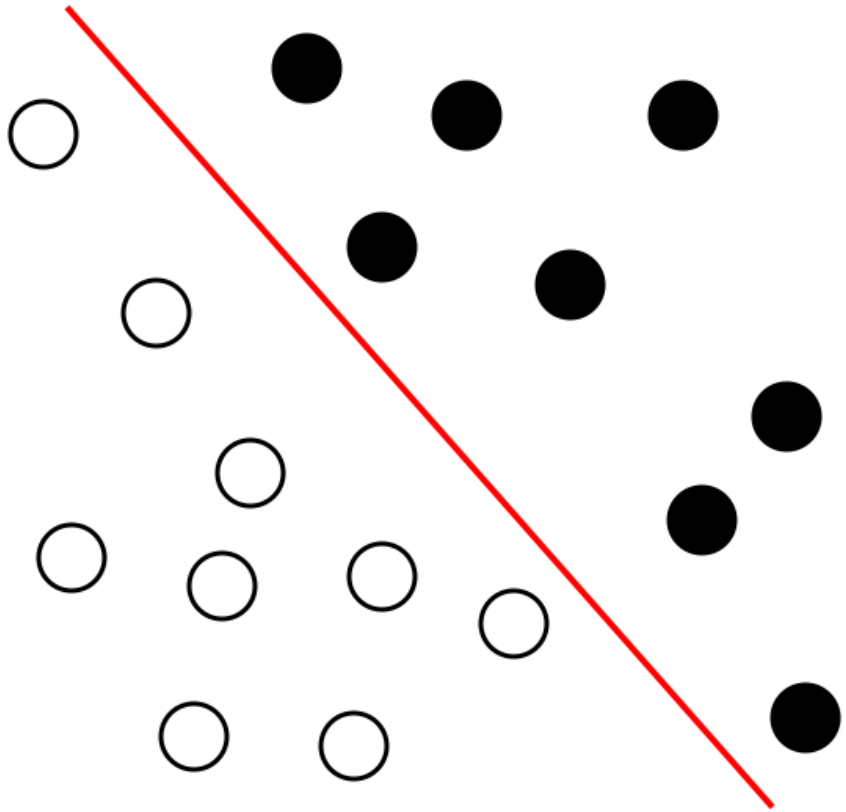
Pattern recognition, *aka* supervised classification



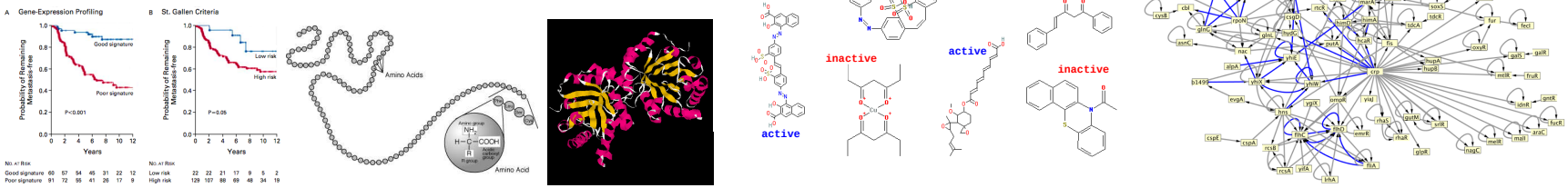


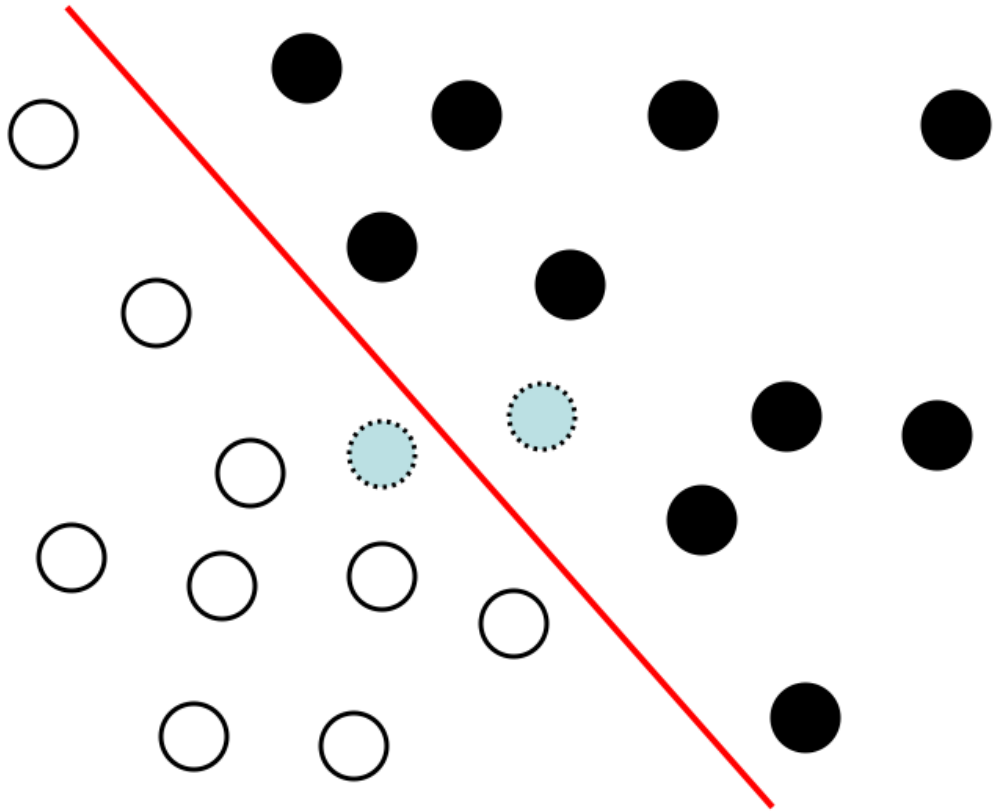
Pattern recognition, *aka* supervised classification



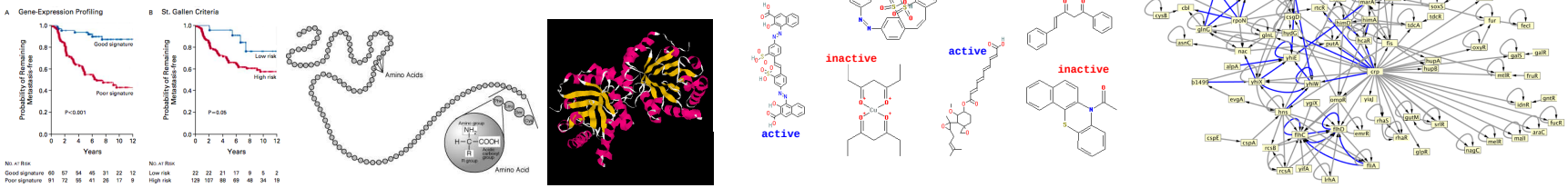


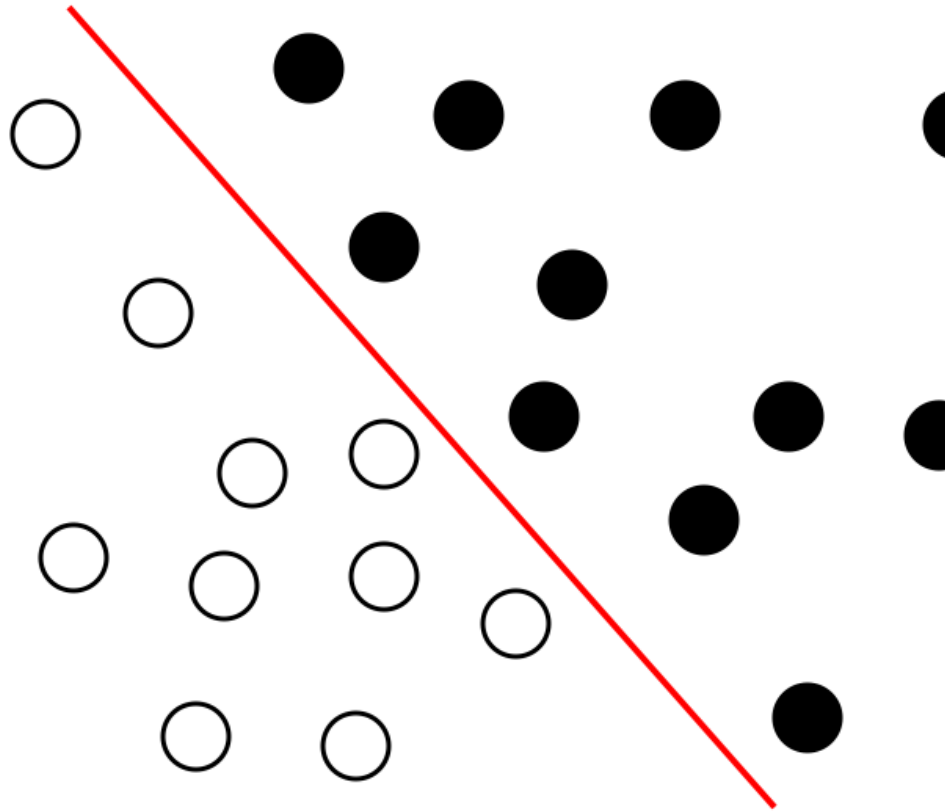
Pattern recognition, *aka* supervised classification



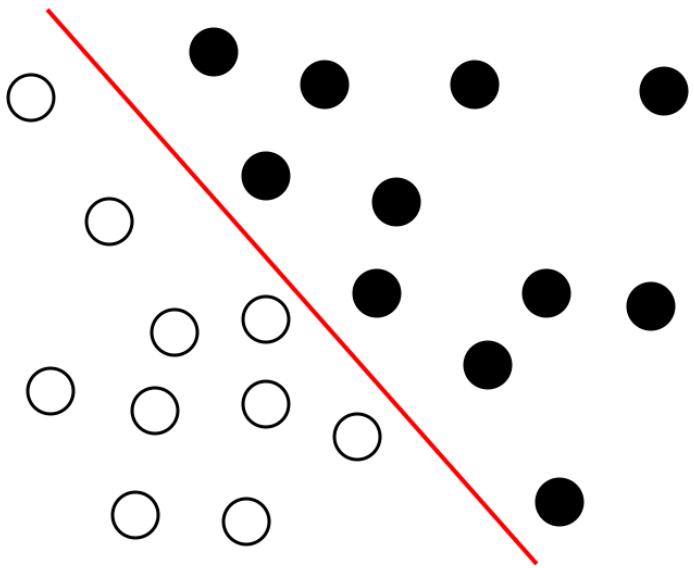


Pattern recognition, *aka* supervised classification





Pattern recognition, *aka* supervised classification



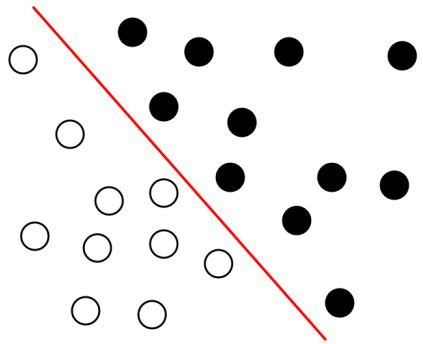
Challenges

- High dimension
- Few samples
- Structured data
- Heterogeneous data
- Prior knowledge
- Fast and scalable implementations
- Interpretable models

Methods for pattern recognitions

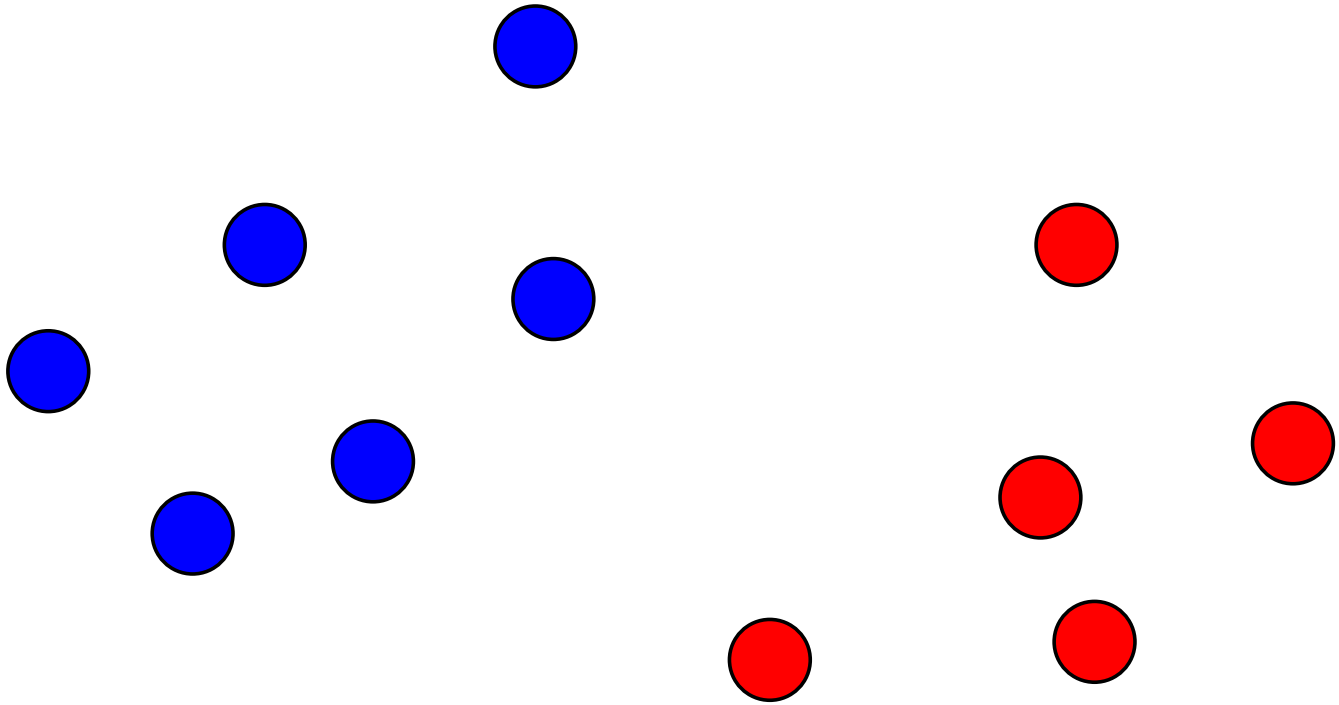
Many methods!

- Logistic regression
- Nearest neighbours
- Decision trees and random forests
- Neural networks
- Support vector machines (SVM)
- ...

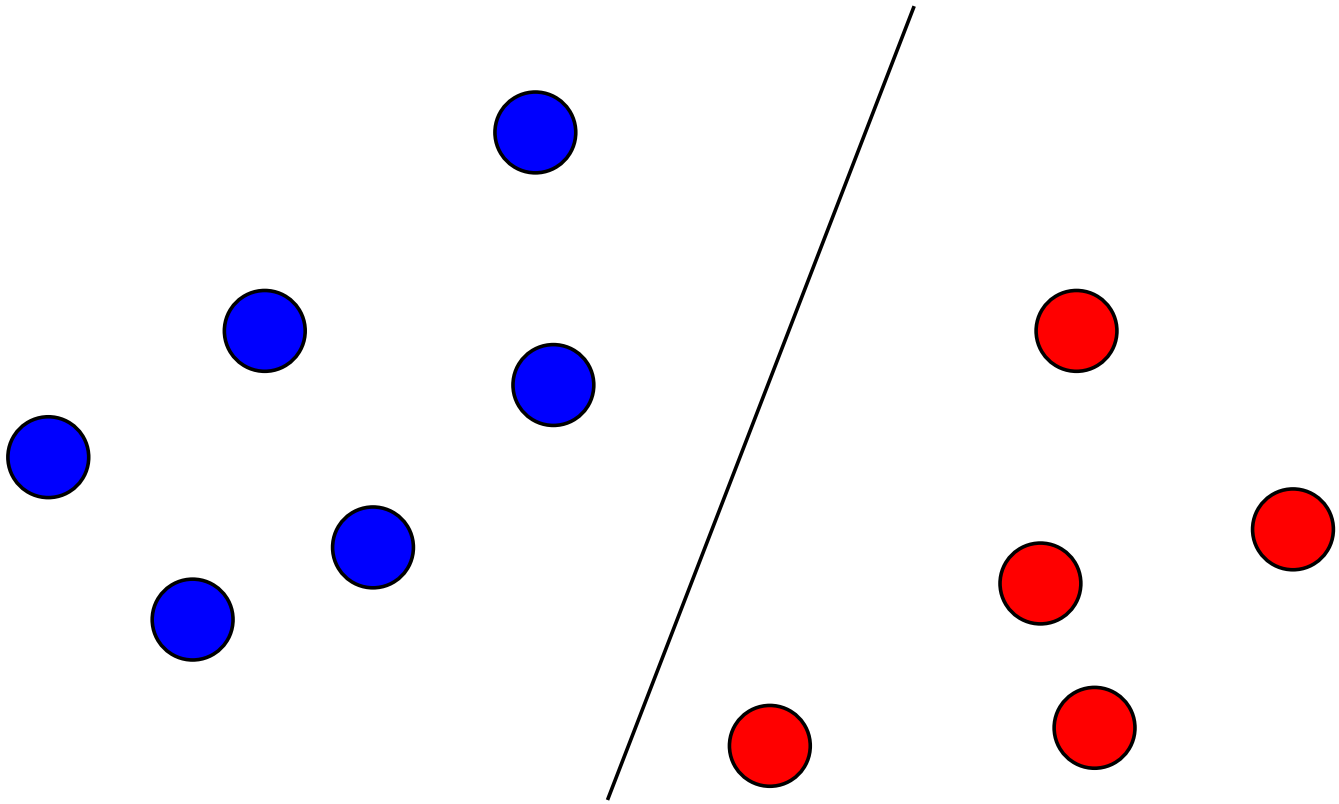


Outline

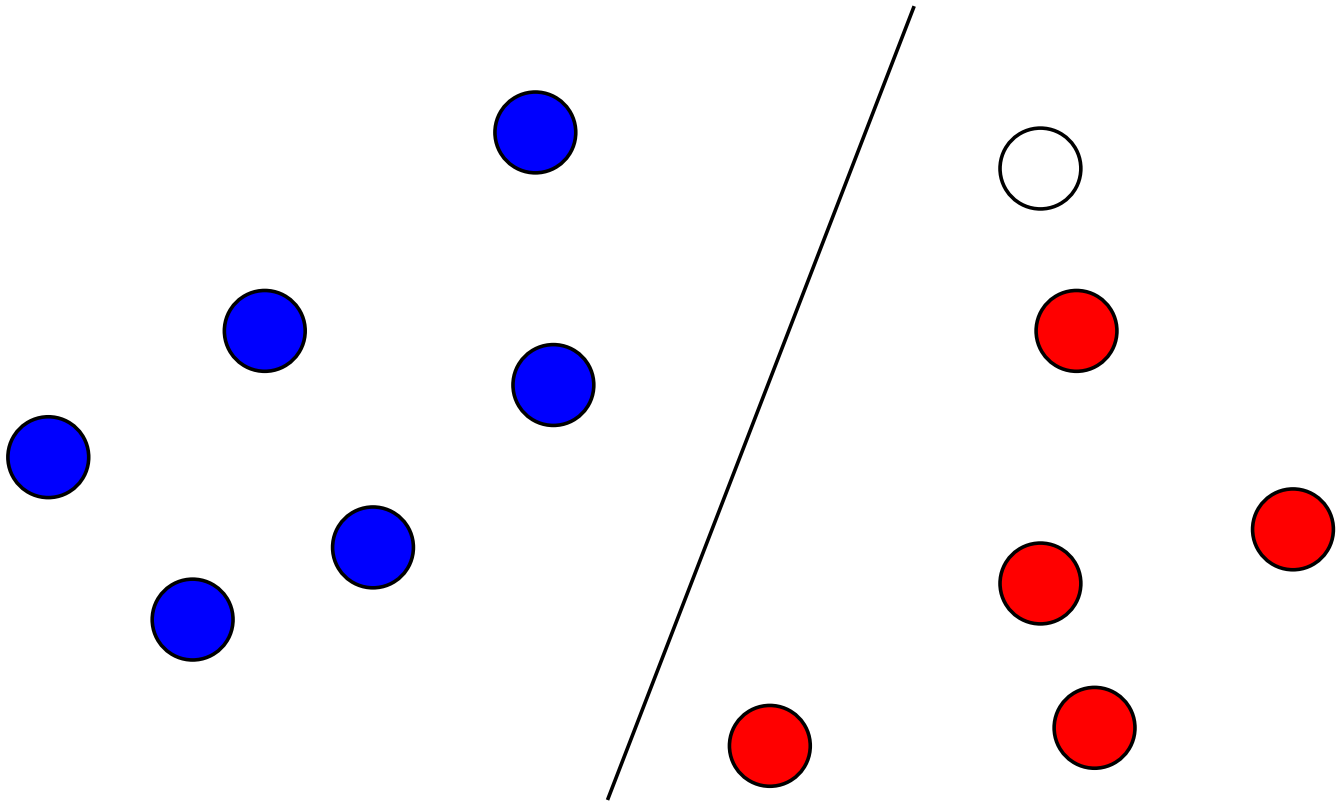
Linear classifiers



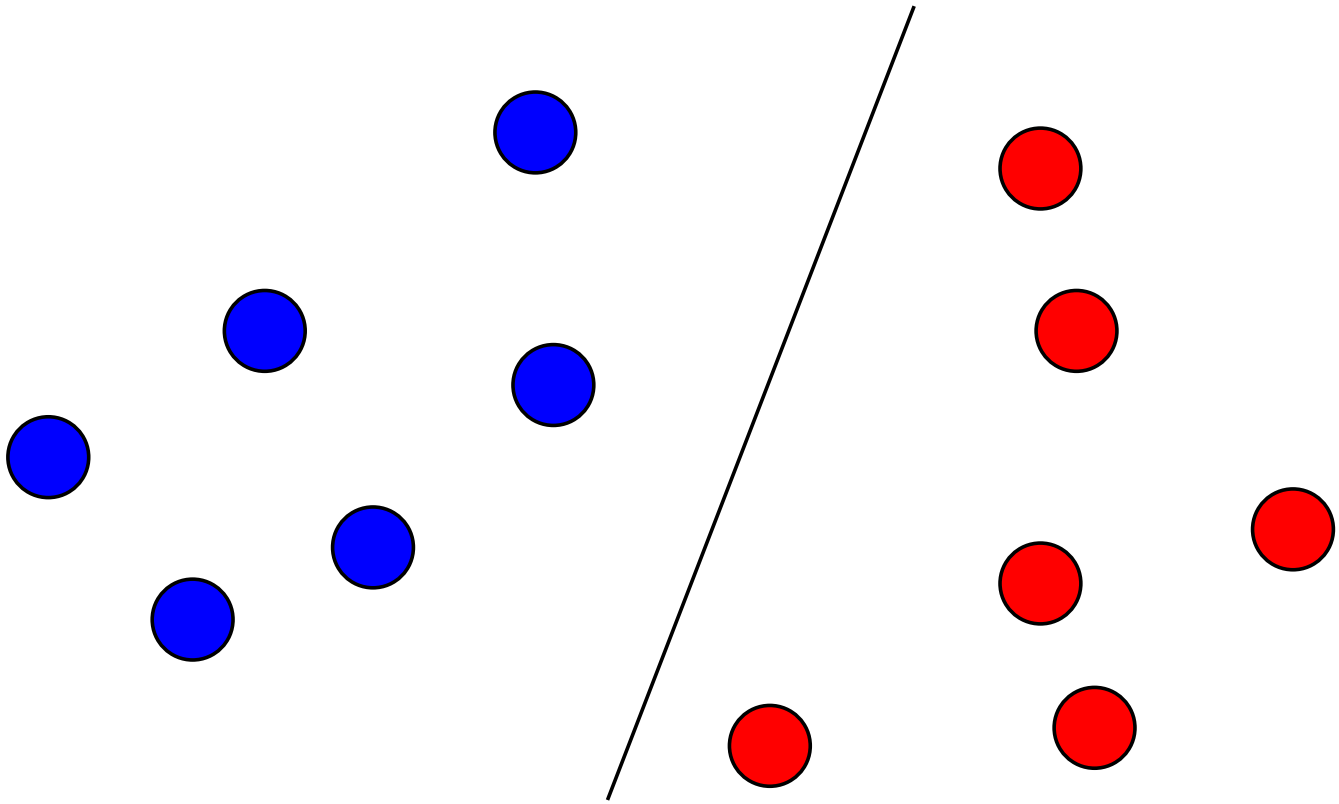
Linear classifiers



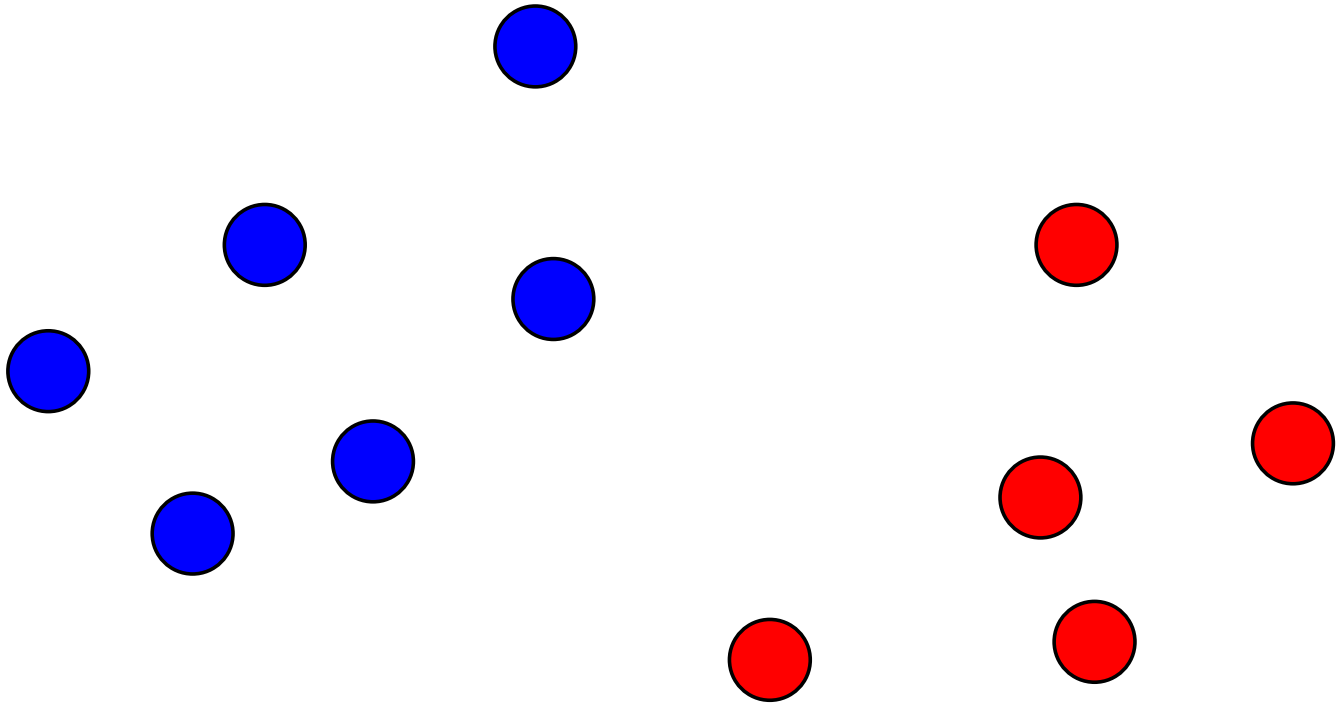
Linear classifiers



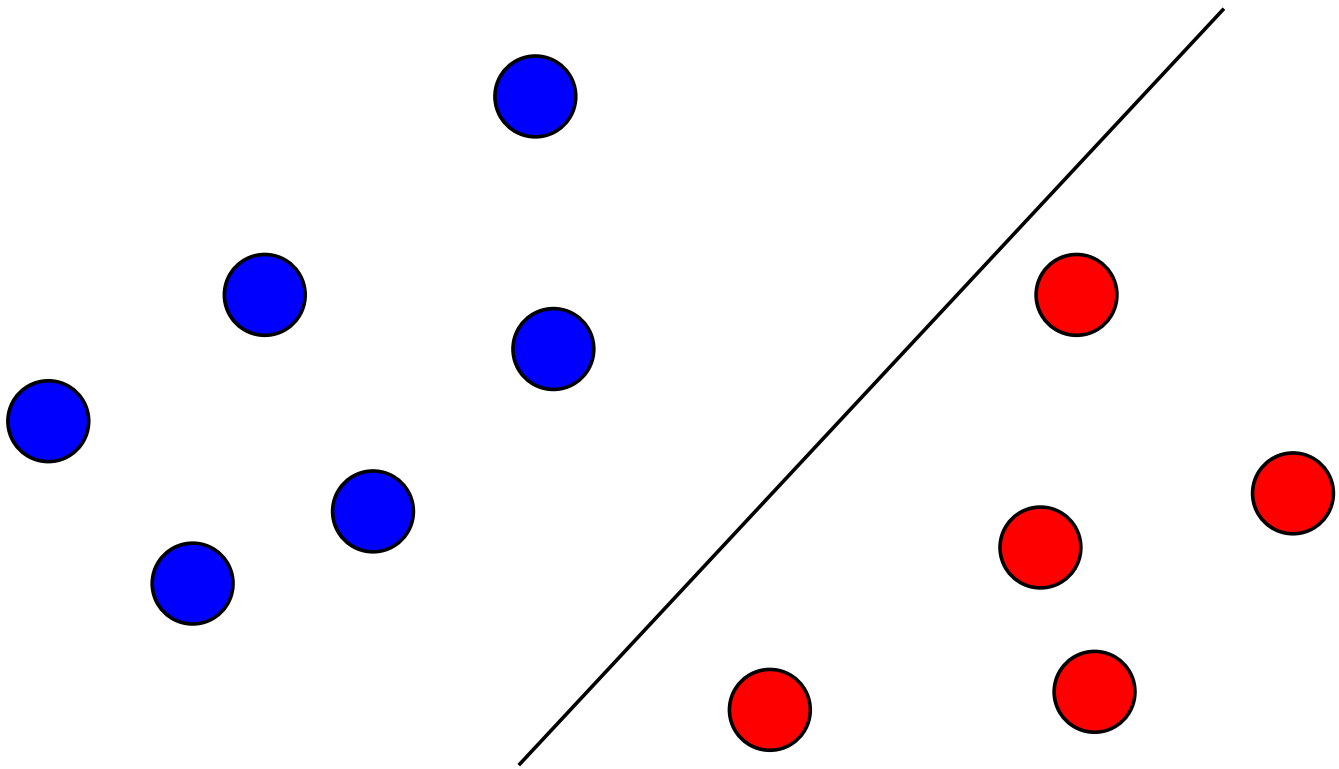
Linear classifiers



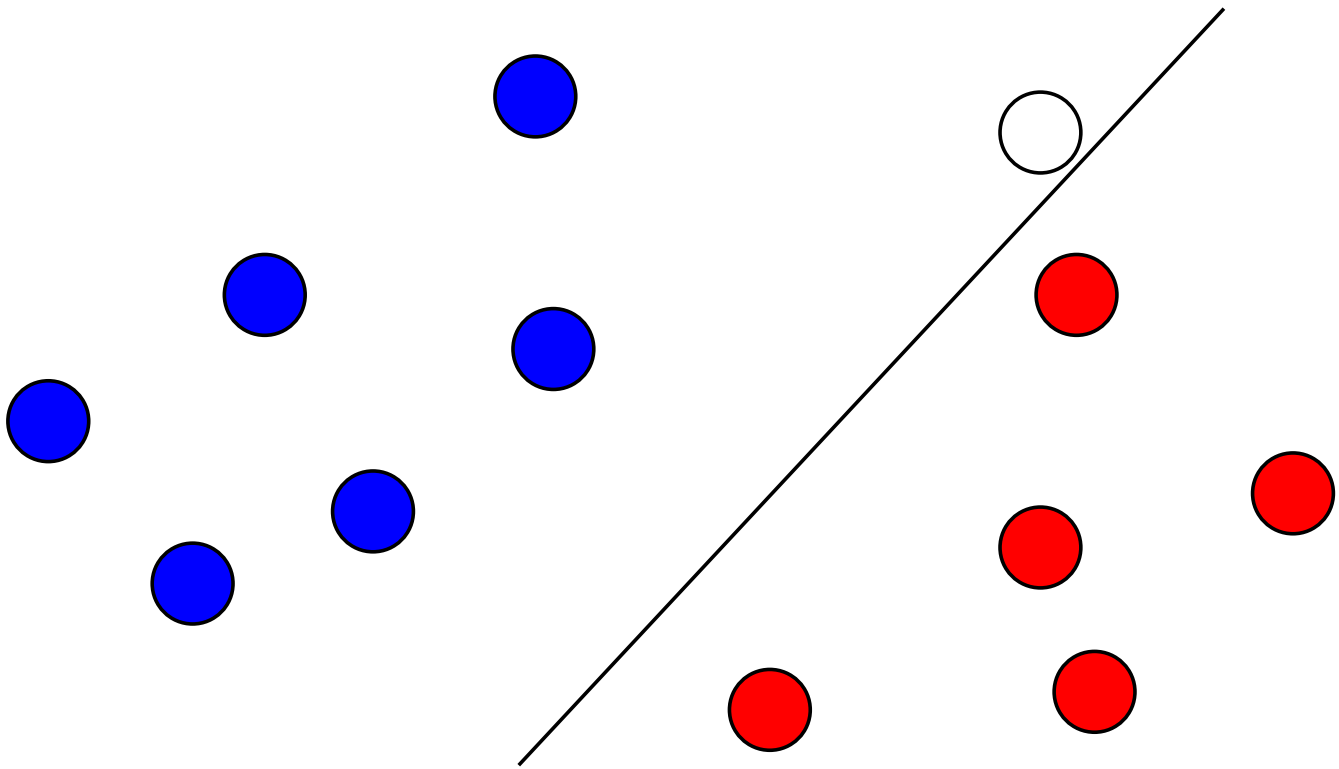
Linear classifiers



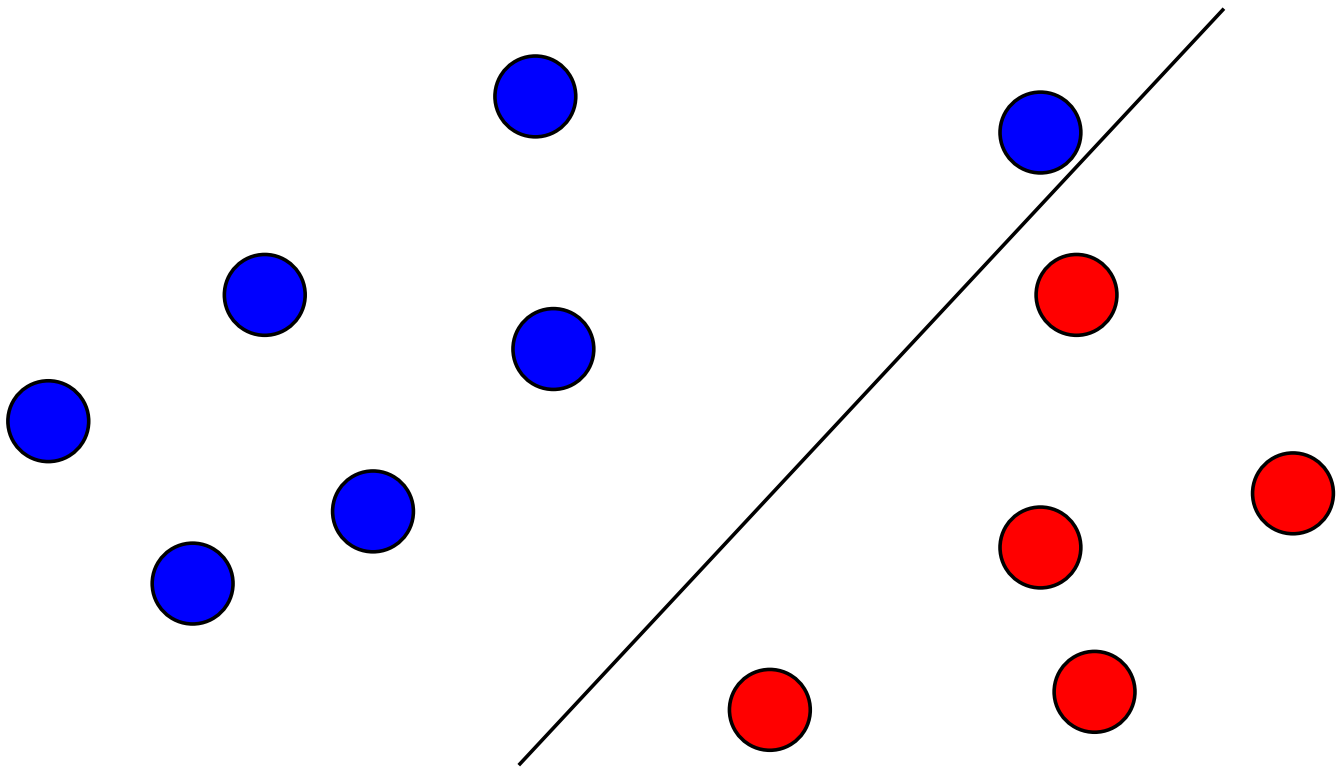
Linear classifiers



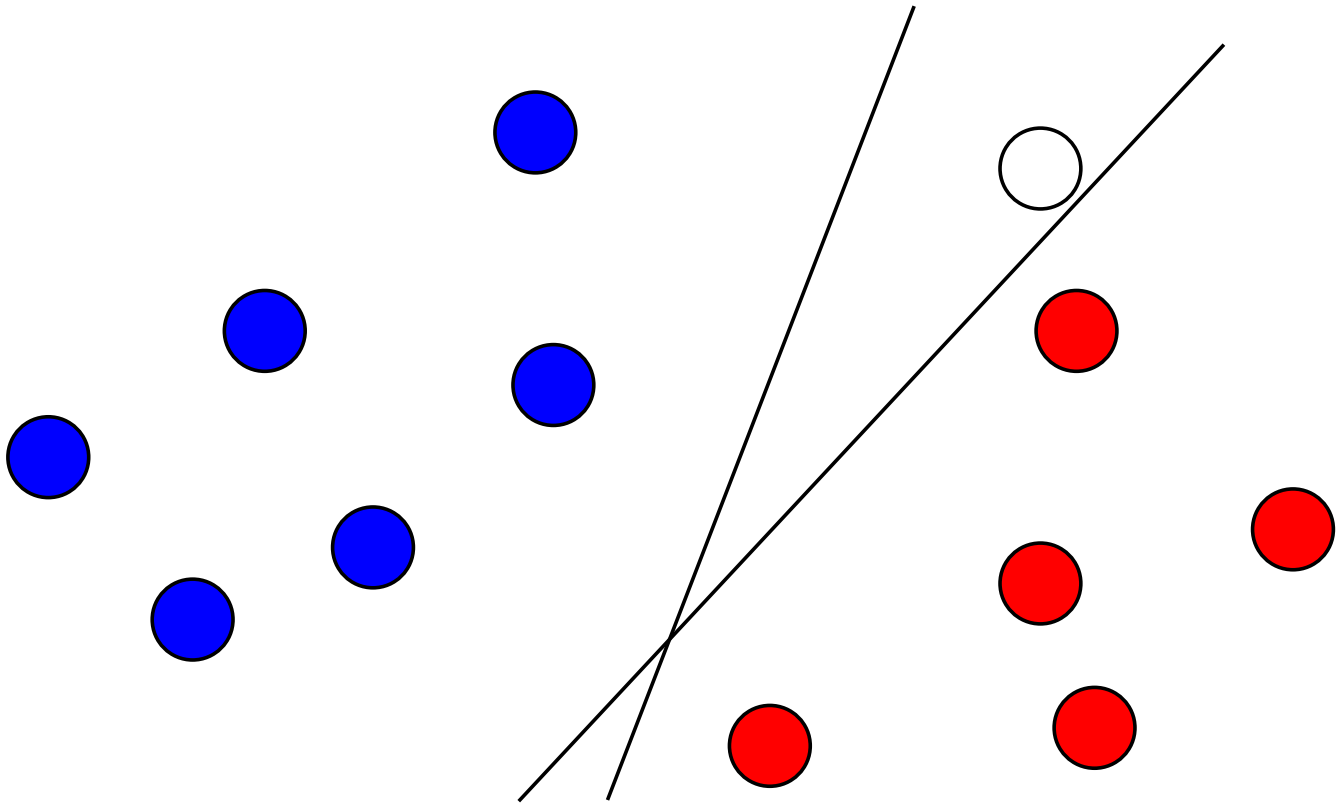
Linear classifiers



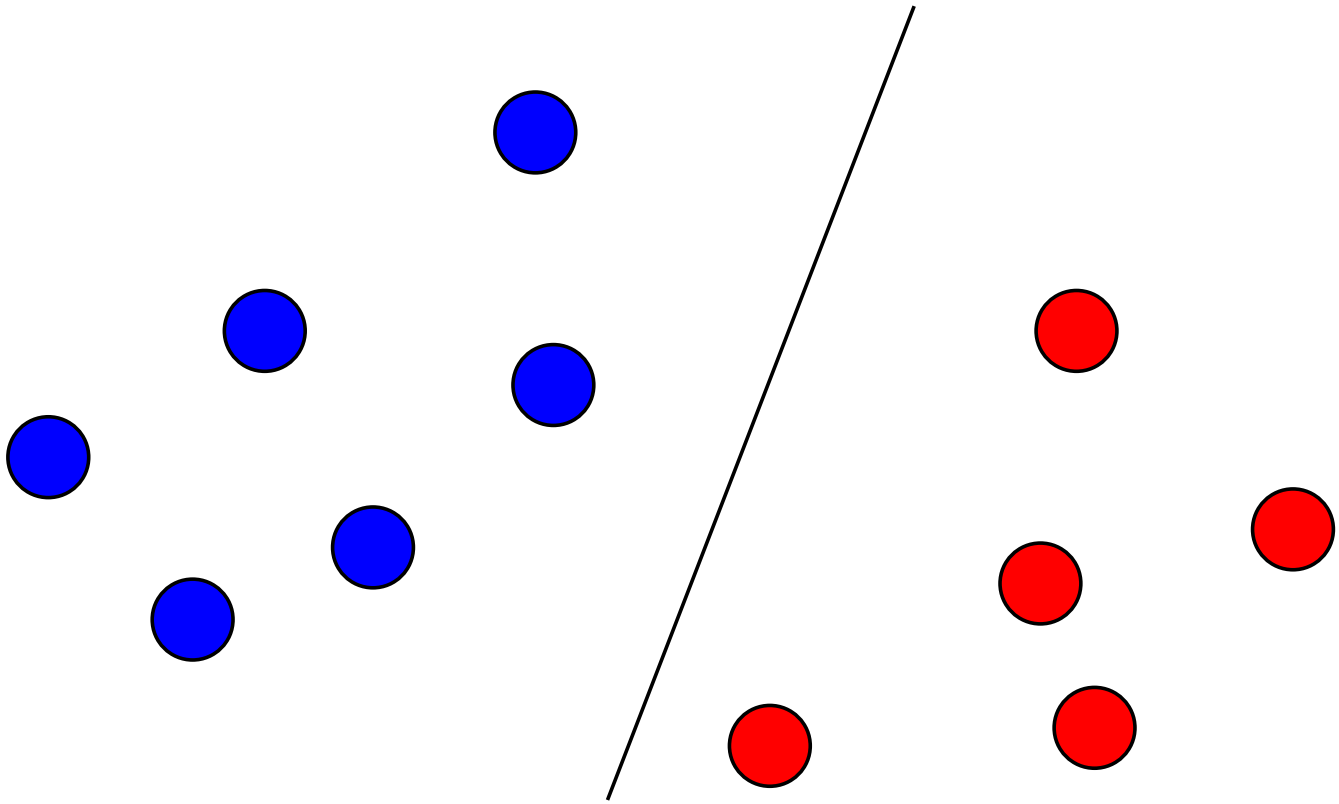
Linear classifiers



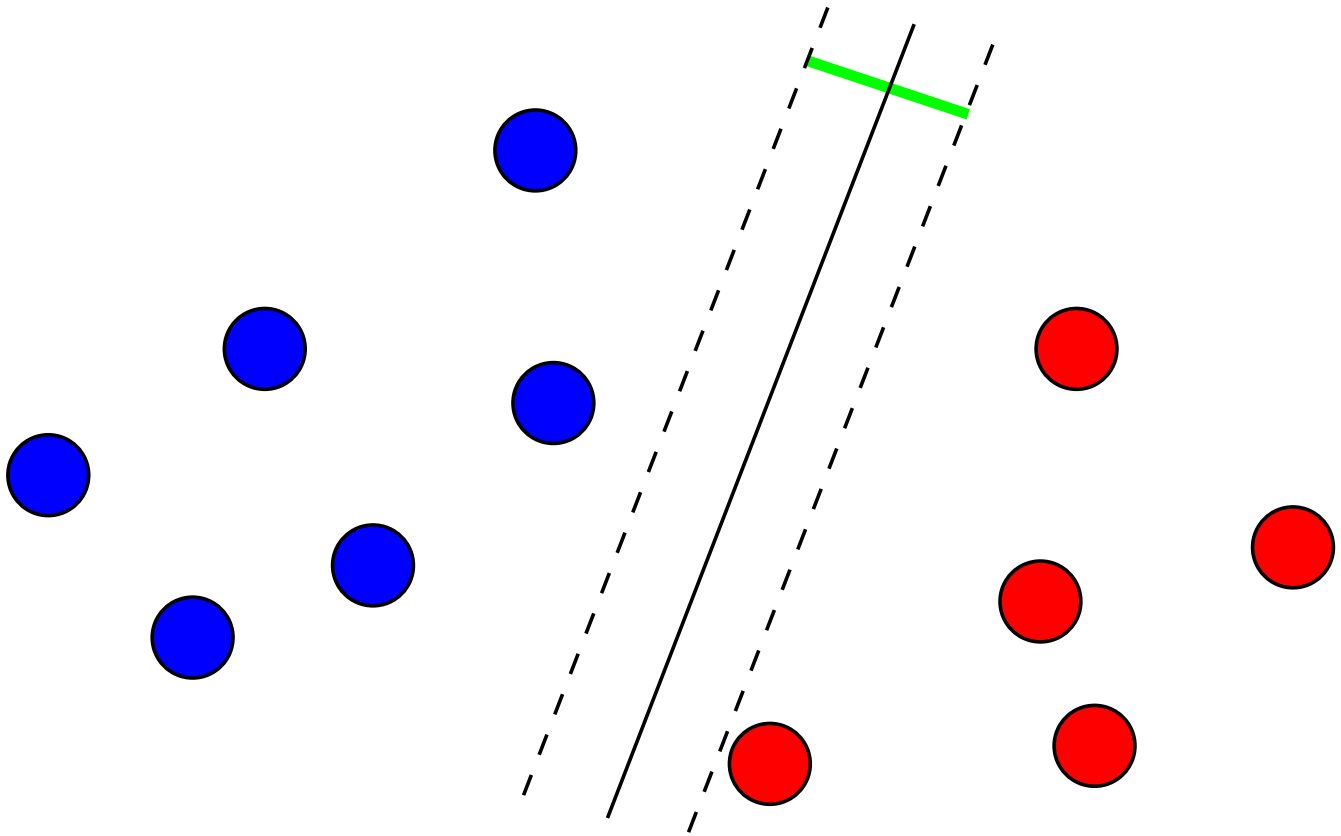
Which one is better?



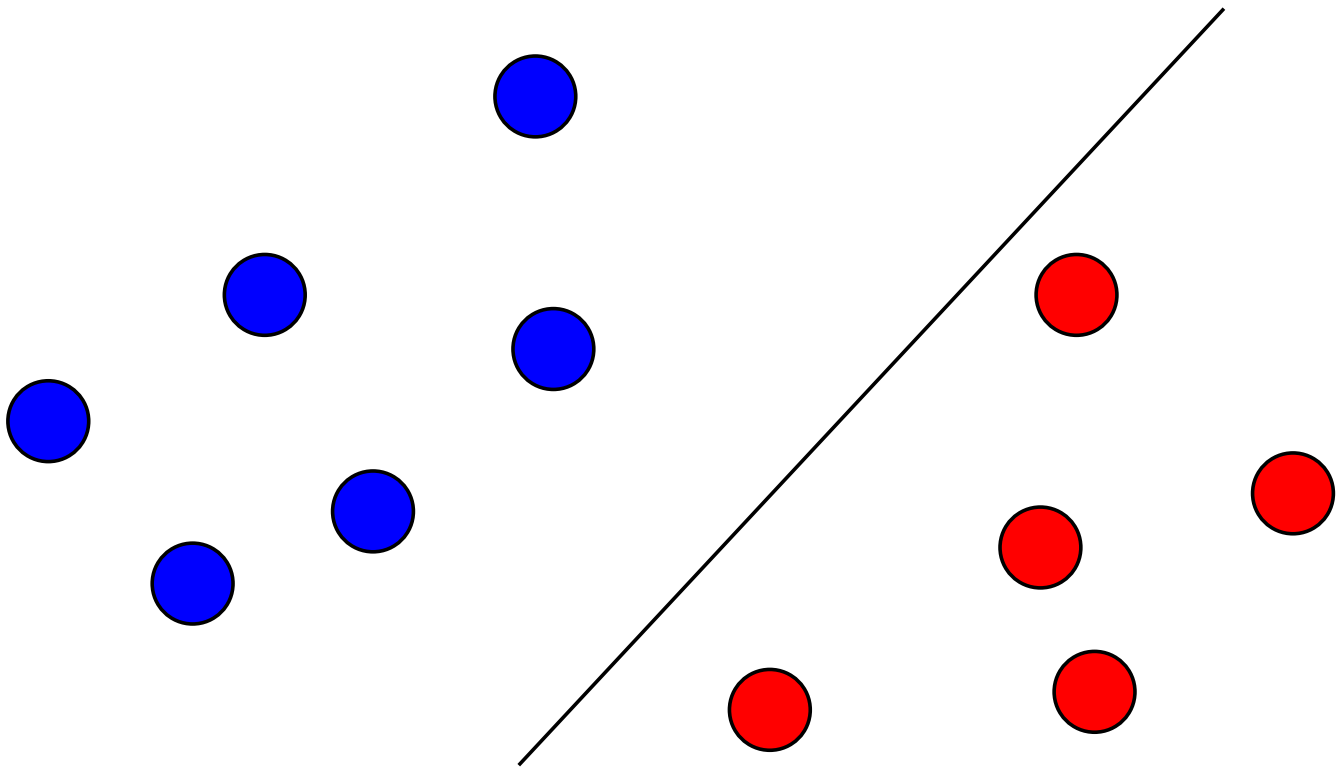
The margin of a linear classifier



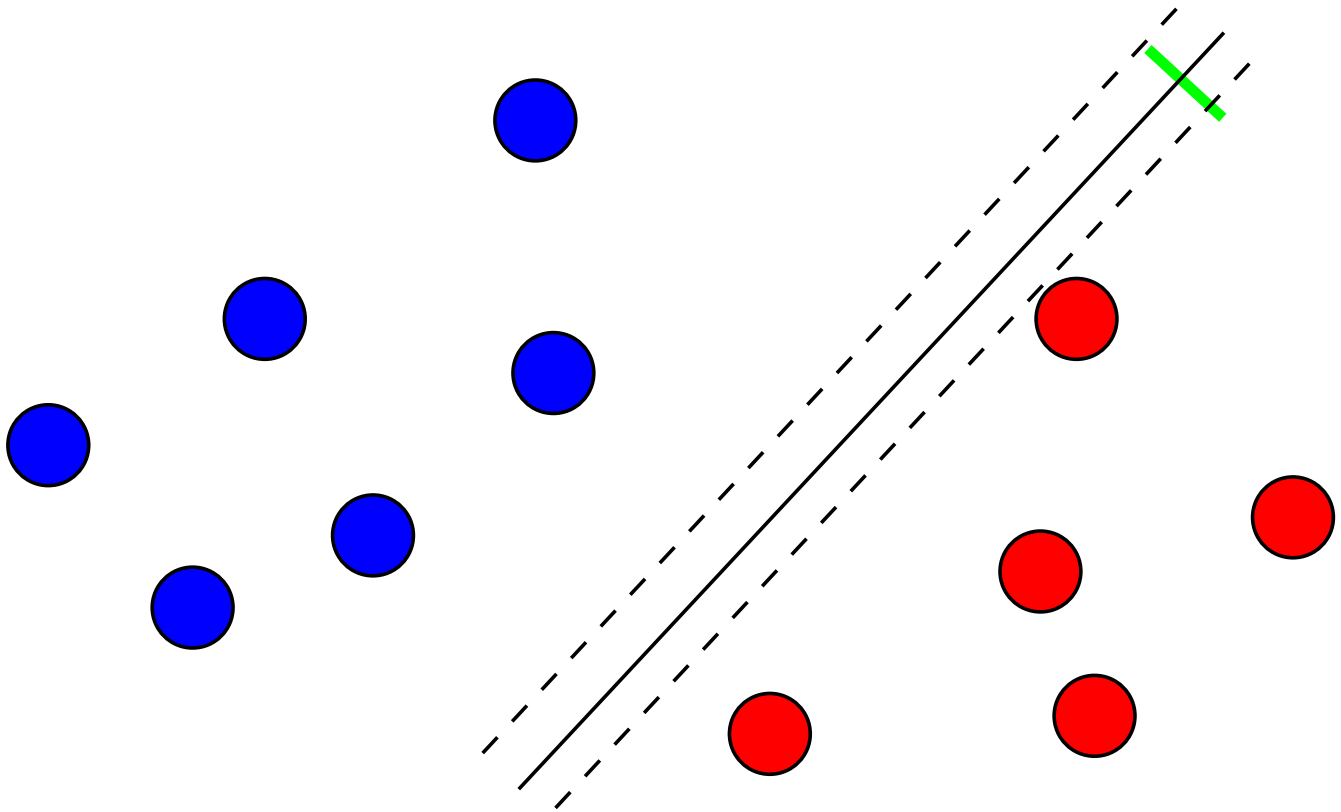
The margin of a linear classifier



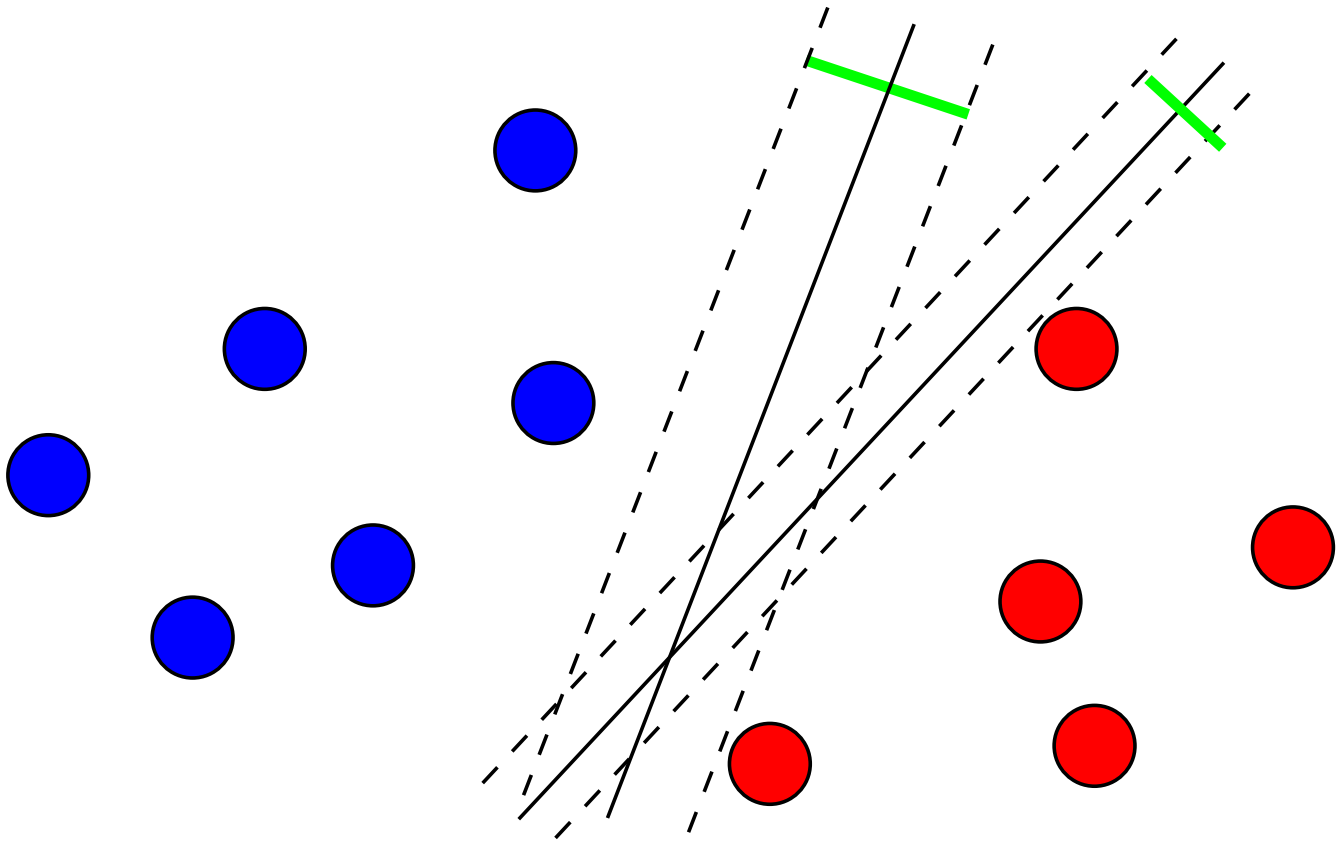
The margin of a linear classifier



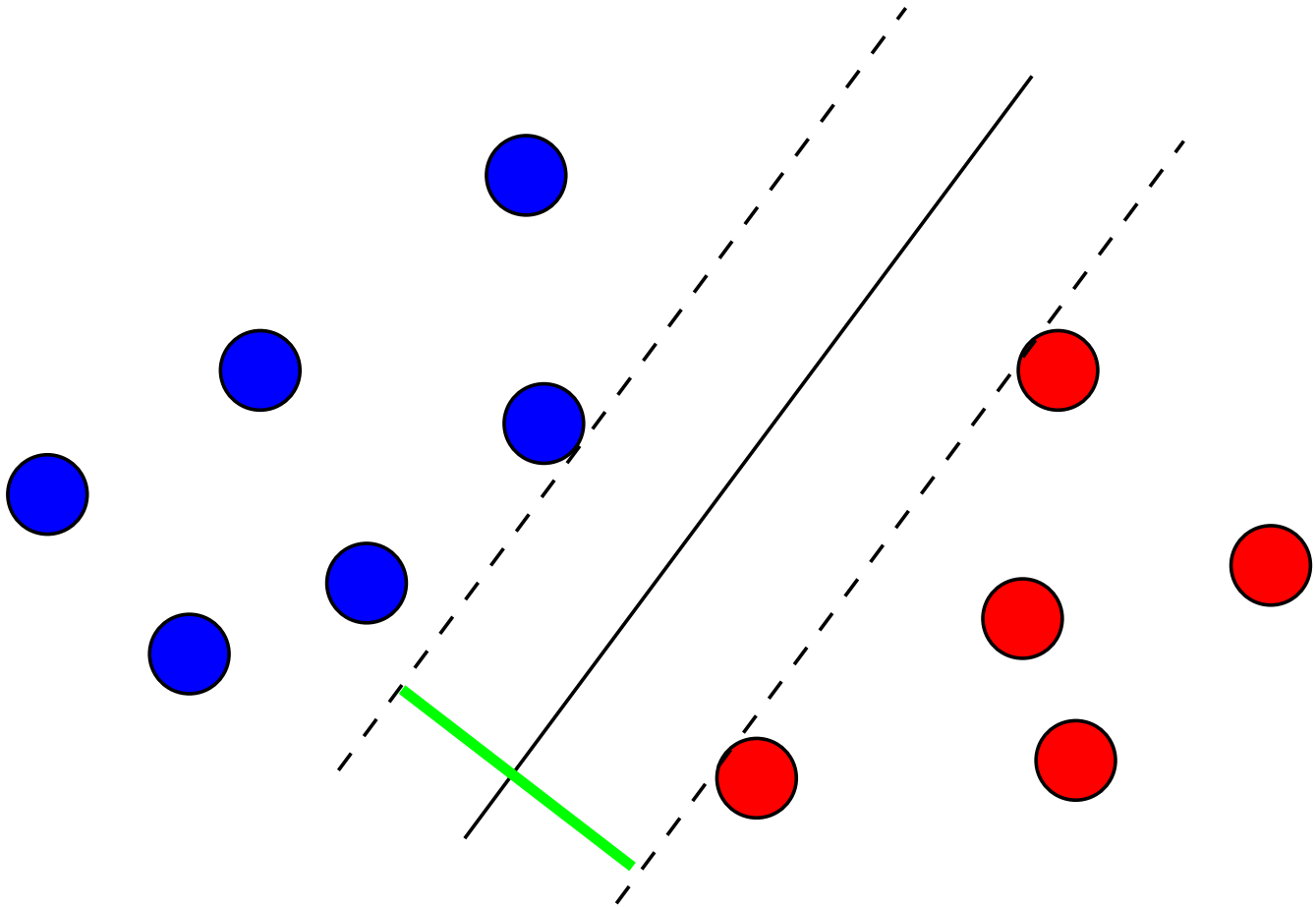
The margin of a linear classifier



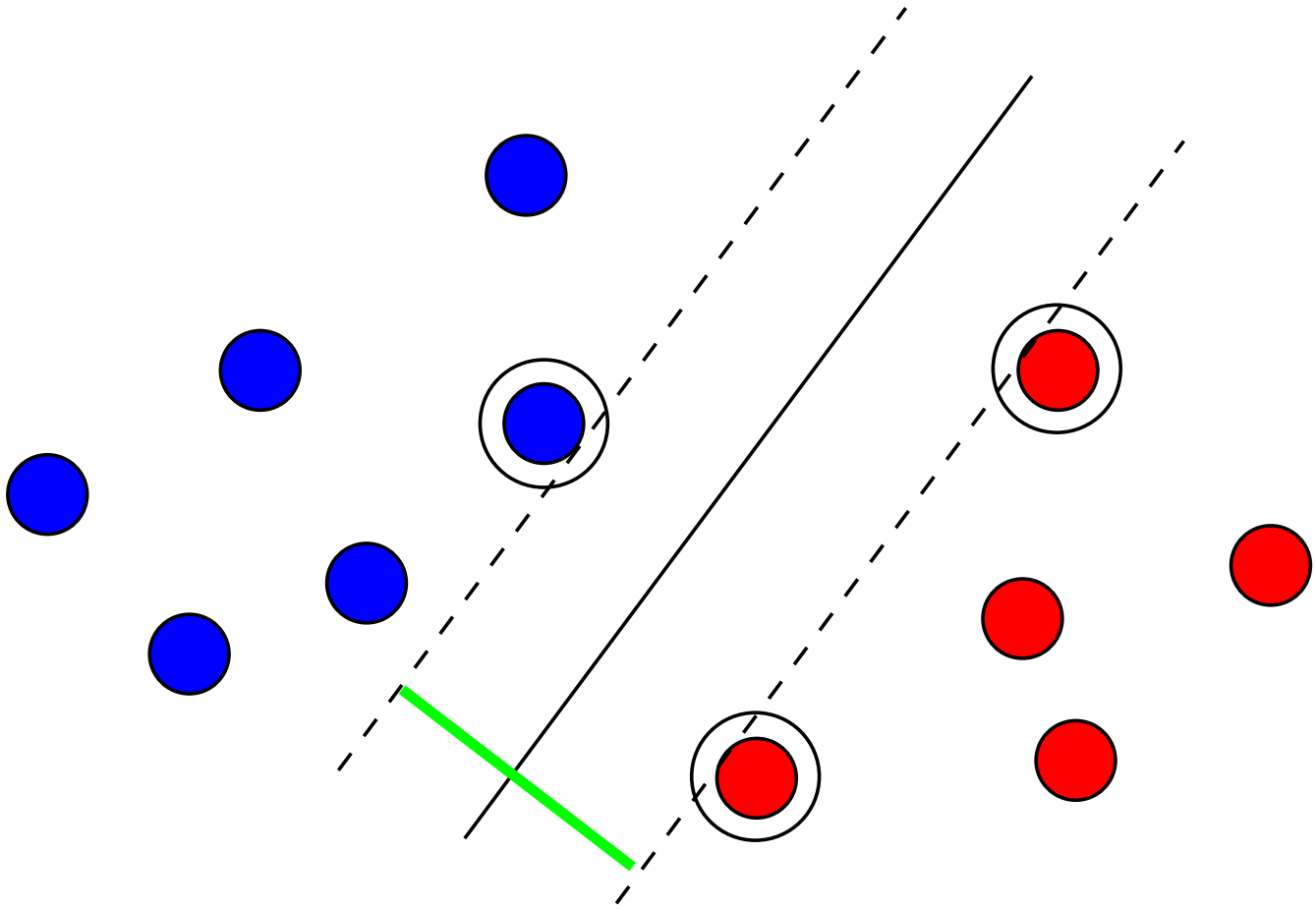
The margin of a linear classifier



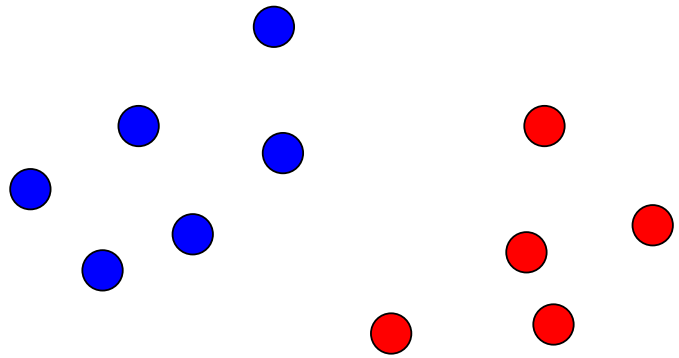
Largest margin classifier (support vector machines)



Support vectors



More formally



- The training set is a finite set of N data/class pairs:

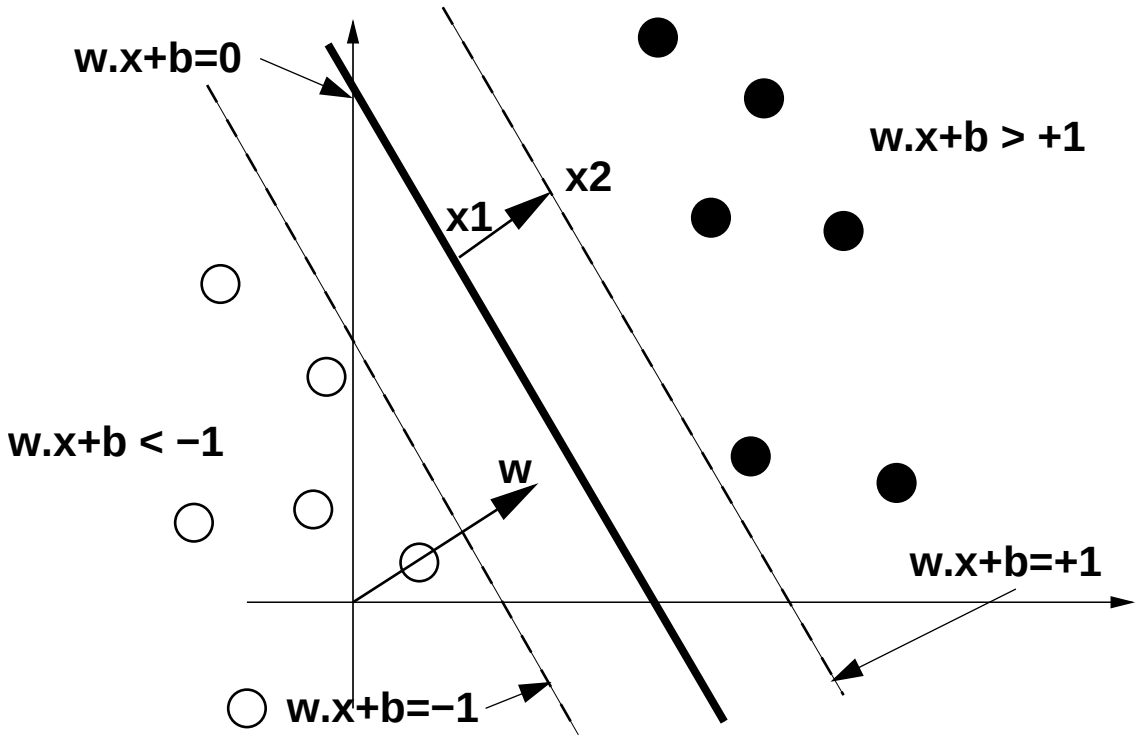
$$\mathcal{S} = \left\{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\right\} ,$$

  where $\vec{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$.
- We assume (for the moment) that the data are linearly separable, i.e., that there exists $(\vec{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ such that:

$$\begin{cases} \vec{w}.\vec{x}_i + b > 0 & \text{if } y_i = 1 , \\ \vec{w}.\vec{x}_i + b < 0 & \text{if } y_i = -1 . \end{cases}$$

How to find the largest separating hyperplane?

For a given linear classifier $f(x) = \vec{w}.\vec{x} + b$ consider the "tube" defined by the values -1 and $+1$ of the decision function:



The margin is $2/||\vec{w}||$

Indeed, the points $\vec{x}_1$ and $\vec{x}_2$ satisfy:

$$\begin{cases} \vec{w}.\vec{x}_1 + b = 0, \\ \vec{w}.\vec{x}_2 + b = 1. \end{cases}$$

By subtracting we get $\vec{w}.(\vec{x}_2 - \vec{x}_1) = 1$, and therefore:

$$\gamma = 2||\vec{x}_2 - \vec{x}_1|| = \frac{2}{||\vec{w}||}.$$

All training points should be on the right side of the dotted line

For positive examples ($y_i = 1$) this means:

$$\vec{w}.\vec{x}_i + b \geq 1$$

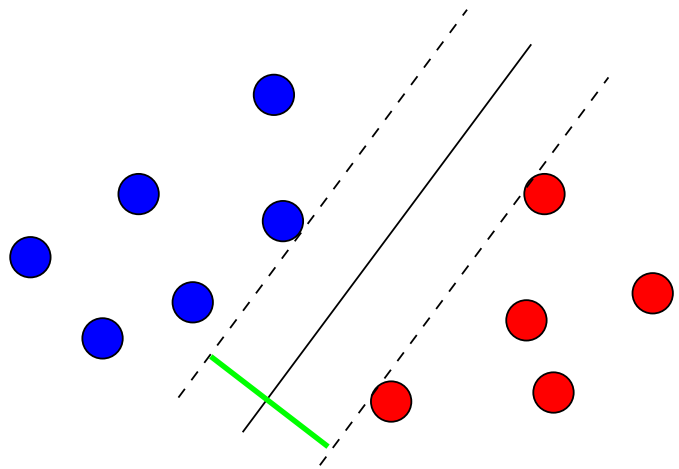
For negative examples ($y_i = -1$) this means:

$$\vec{w}.\vec{x}_i + b \leq -1$$

Both cases are summarized by:

$$\forall i = 1, \ldots, N, \qquad y_i \left(\vec{w}.\vec{x}_i + b\right) \geq 1$$

Finding the optimal hyperplane



Find $(\vec{w}, b)$ which minimize:

$$||\vec{w}||^2$$

under the constraints:

$$\forall i = 1, \ldots, N, \qquad y_i \left(\vec{w}.\vec{x}_i + b\right) - 1 \geq 0.$$

This is a classical quadratic program on $\mathbb{R}^{d+1}$.

Lagrangian

In order to minimize:

$$\frac{1}{2}||\vec{w}||^2$$

under the constraints:

$$\forall i = 1, \ldots, N, \qquad y_i \left(\vec{w}.\vec{x}_i + b\right) - 1 \geq 0.$$

we introduce one dual variable α_i for each constraint, i.e., for each training point. The Lagrangian is:

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{N} \alpha_i \left(y_i \left(\vec{w}.\vec{x}_i + b\right) - 1\right).$$

Dual problem

Find $\alpha^* \in \mathbb{R}^N$ which maximizes

$$L(\vec{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \vec{x}_i . \vec{x}_j,$$

under the (simple) constraints $\alpha_i \geq 0$ (for $i = 1, \ldots, N$), and

$$\sum_{i=1}^{N} \alpha_i y_i = 0.$$

This is a quadratic program on $\mathbb{R}^N$, with "box constraints". $\vec{\alpha}^*$ can be found efficiently using dedicated optimization softwares.

Recovering the optimal hyperplane

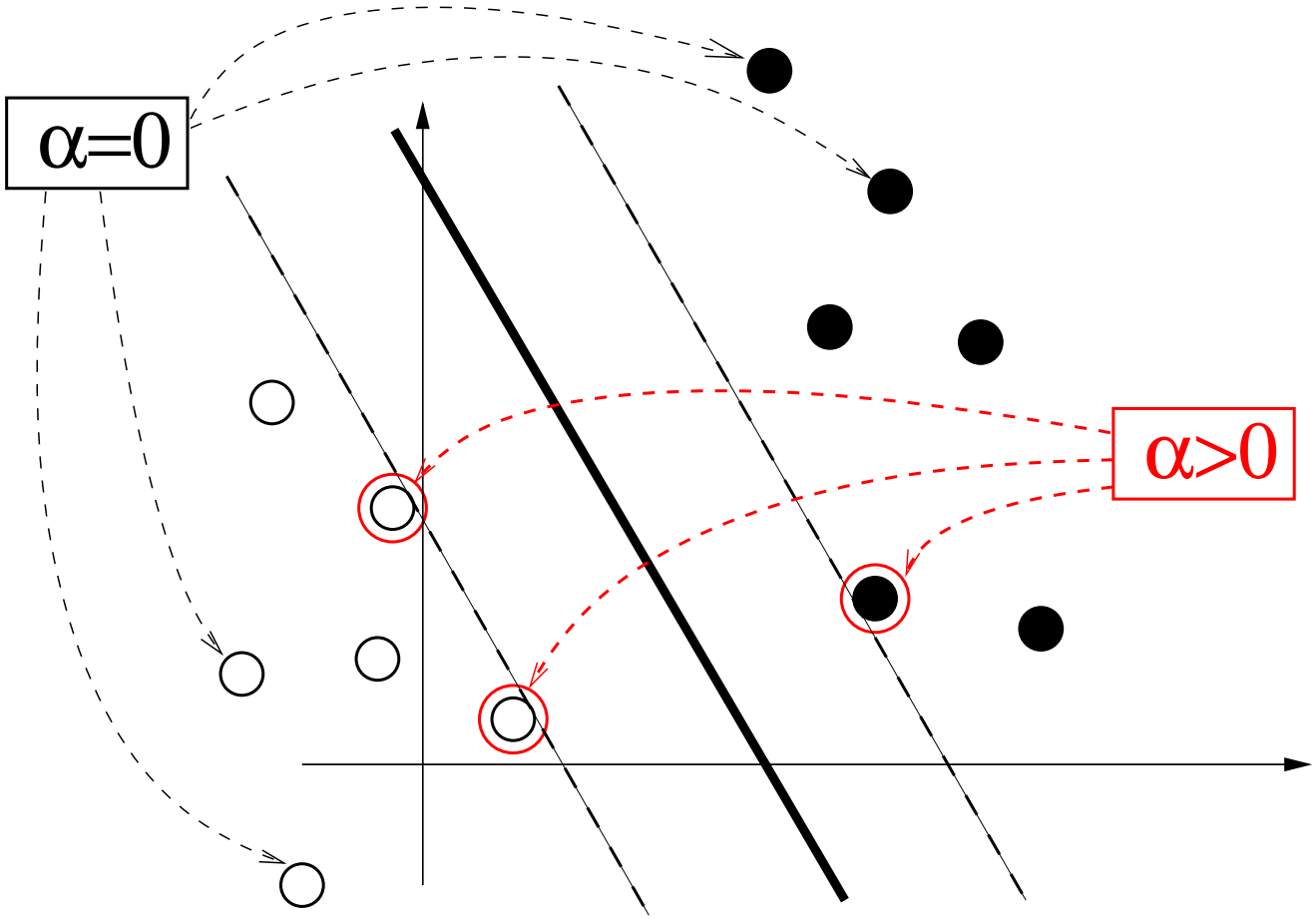
Once $\vec{\alpha}^*$ is found, we recover $(\vec{w}^*, b^*)$ corresponding to the optimal hyperplane. w^* is given by:

$$\vec{w}^* = \sum_{i=1}^{N} \alpha_i \vec{x}_i,$$

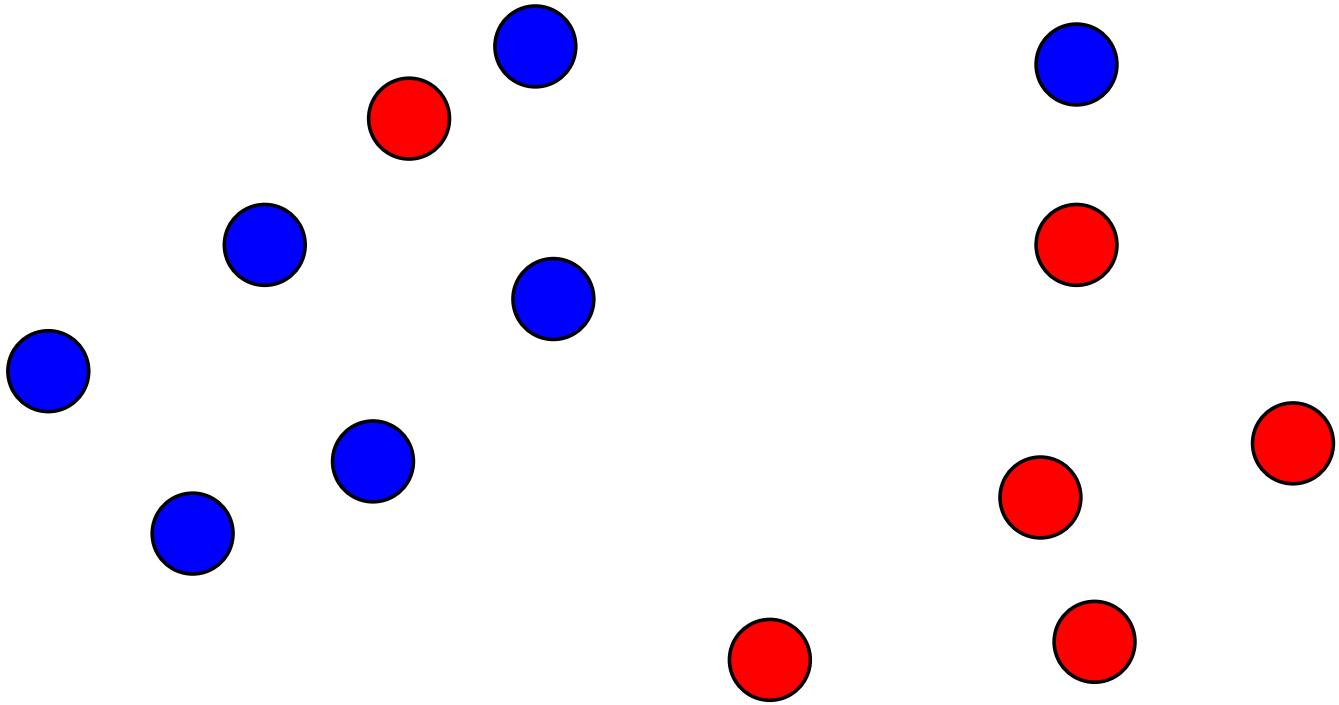
and the decision function is therefore:

$$\begin{aligned} f^*(\vec{x}) &= \vec{w}^*.\vec{x} + b^* \\ &= \sum_{i=1}^{N} \alpha_i \vec{x}_i.\vec{x} + b^*. \end{aligned} \tag{1}$$

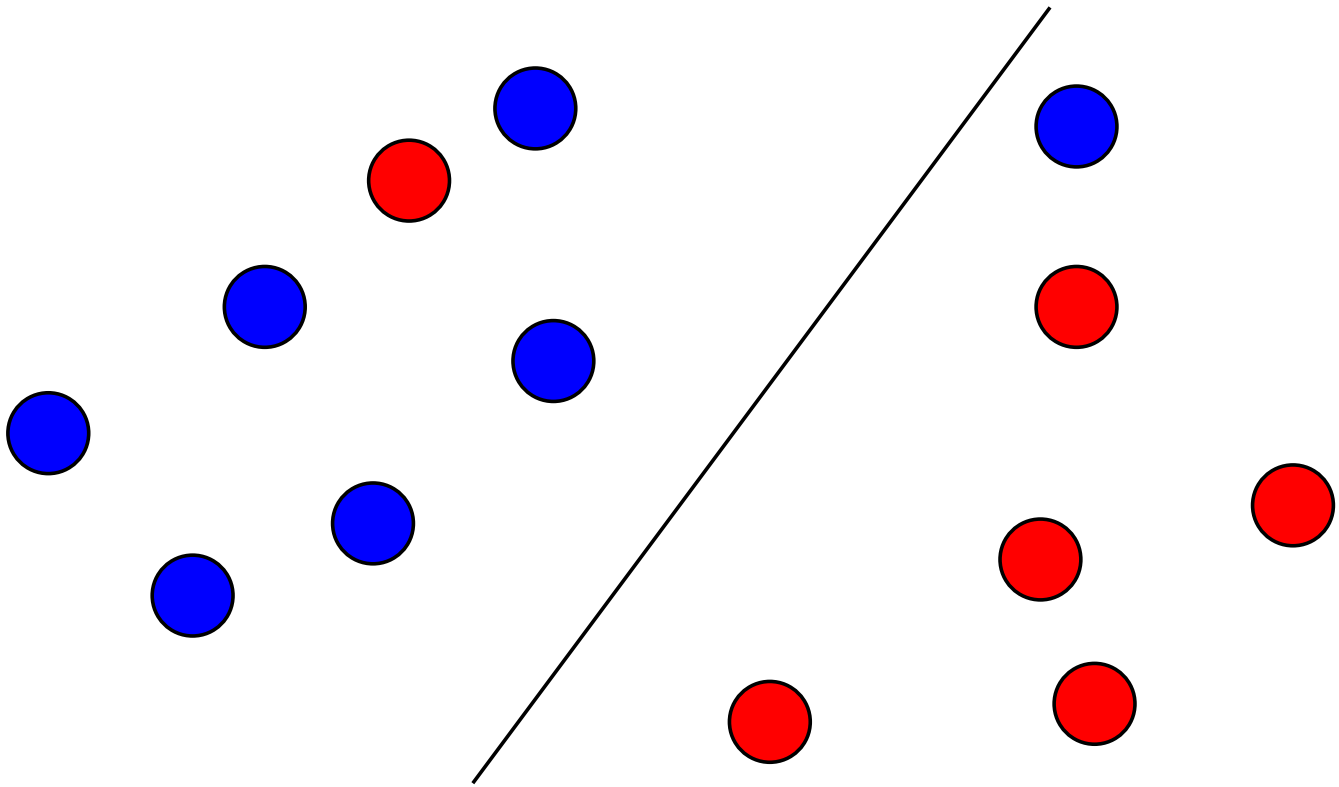
Interpretation: support vectors



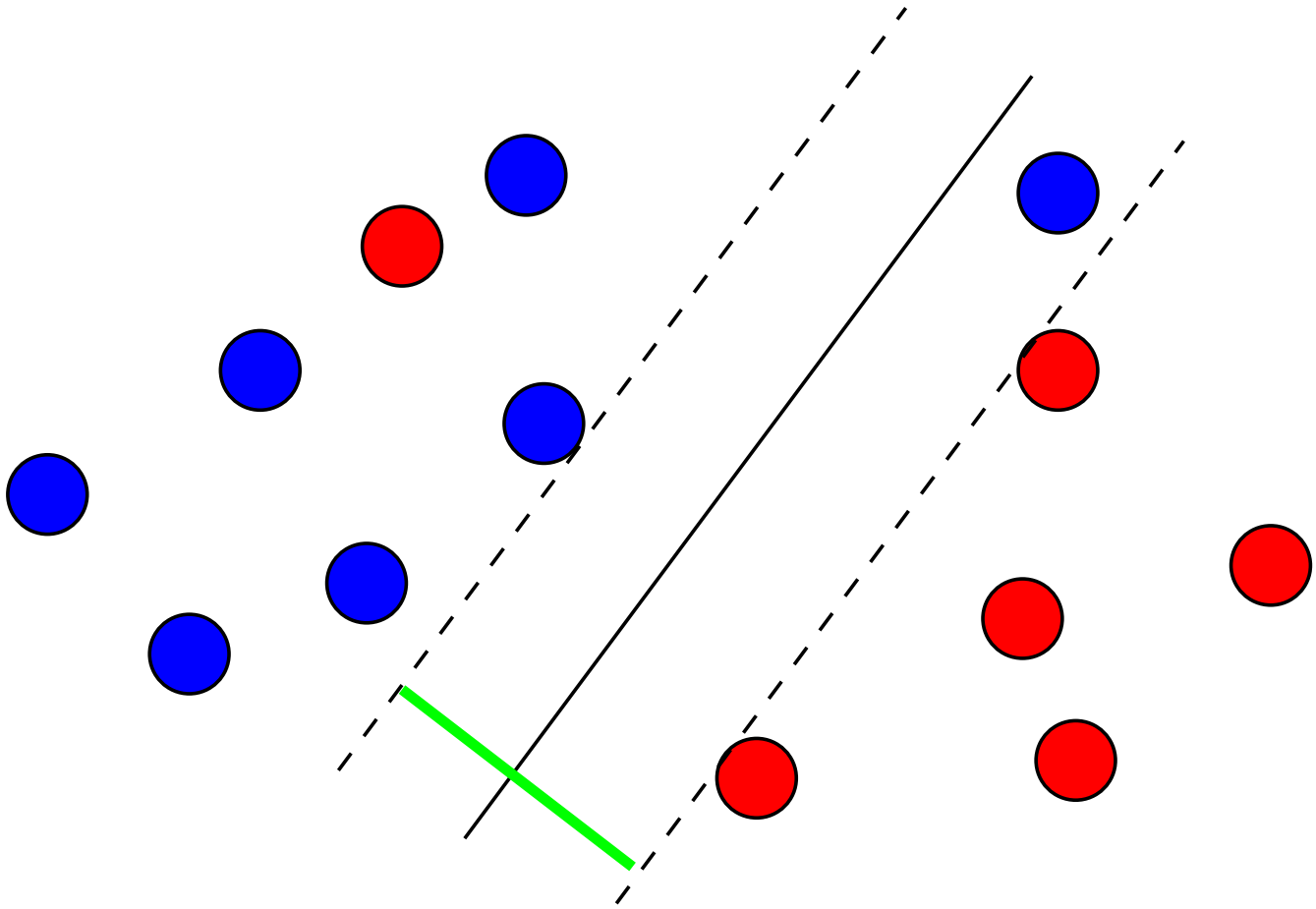
What if data are not linearly separable?



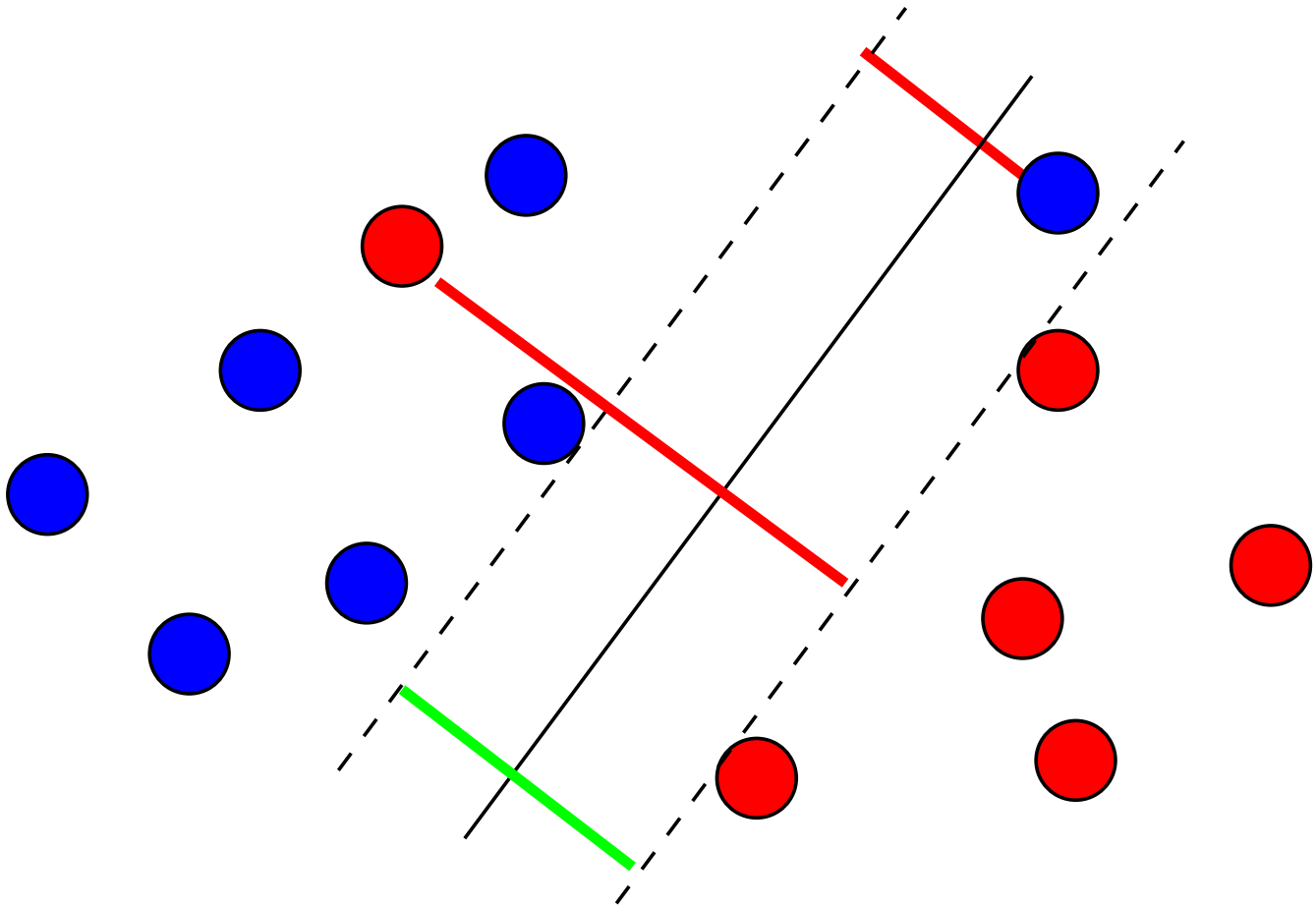
What if data are not linearly separable?



What if data are not linearly separable?



What if data are not linearly separable?

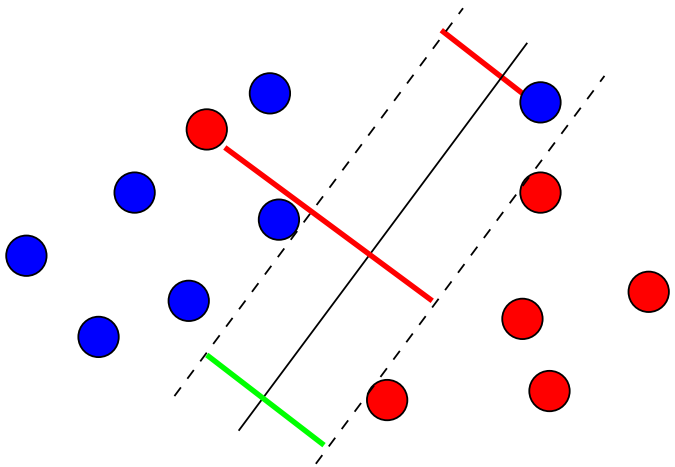


Soft-margin SVM

- Find a trade-off between large margin and few errors.
- Mathematically:

$$\min_f \left\{ \frac{1}{margin(f)} + C \times errors(f) \right\}$$

- C is a parameter



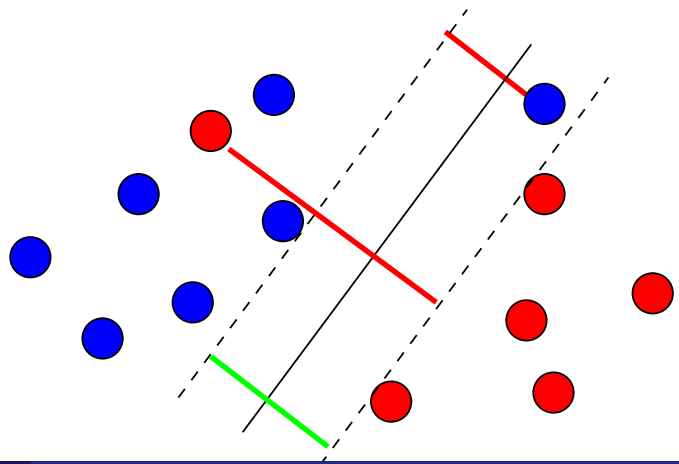
Soft-margin SVM formulation

- The margin of a labeled point $(\vec{x}, y)$ is

$$margin(\vec{x}, y) = y\left(\vec{w}.\vec{x} + b\right)$$

- The error is
  - 0 if $margin(\vec{x}, y) > 1$,
  - $1 - margin(\vec{x}, y)$ otherwise.
- The soft margin SVM solves:

$$\min_{\vec{w},b} \left\{ ||\vec{w}||^2 + C \sum_{i=1}^{N} \max\left(0, 1 - y_i\left(\vec{w}.\vec{x}_i + b\right)\right) \right\}$$



Dual formulation of soft-margin SVM

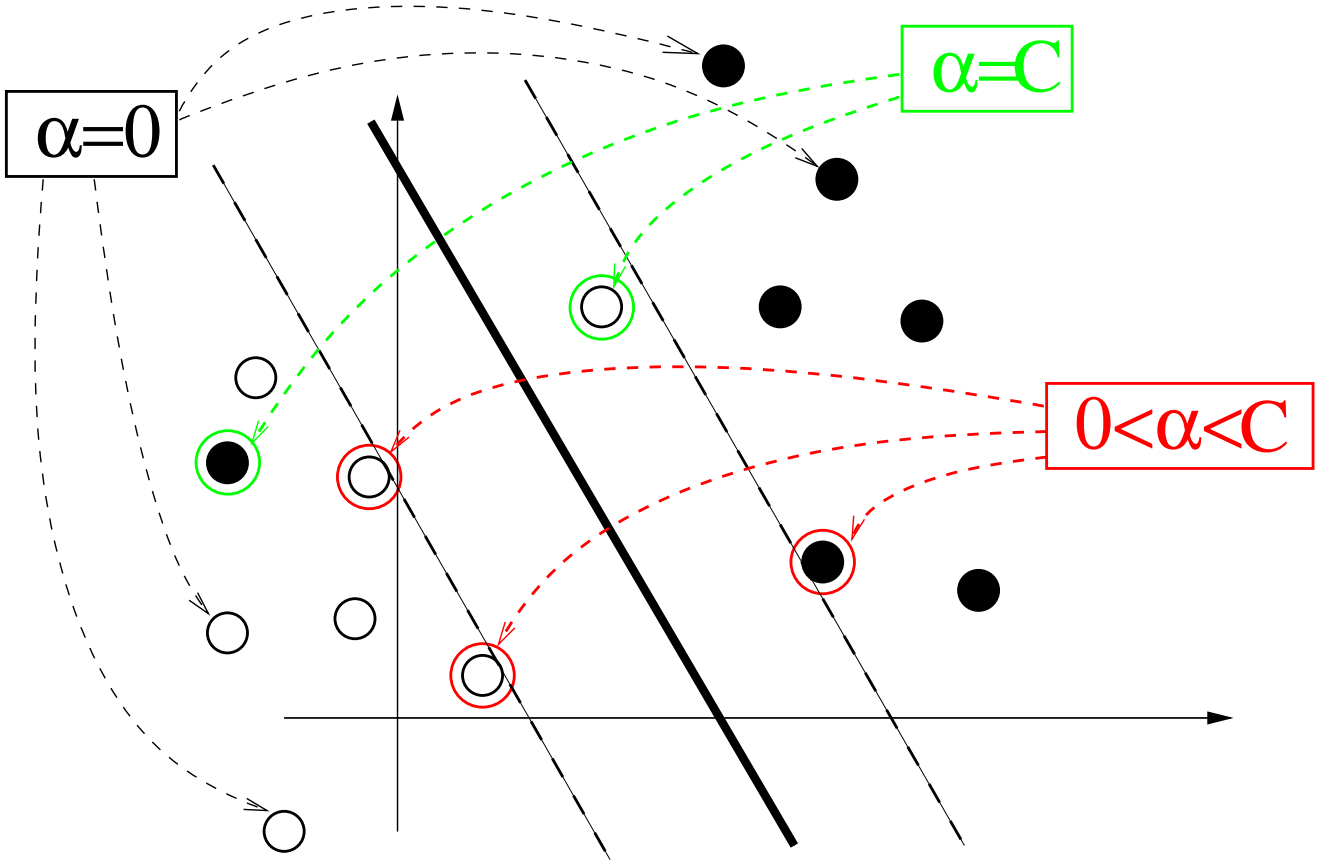
Maximize

$$L(\vec{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \vec{x}_i . \vec{x}_j,$$

under the constraints:

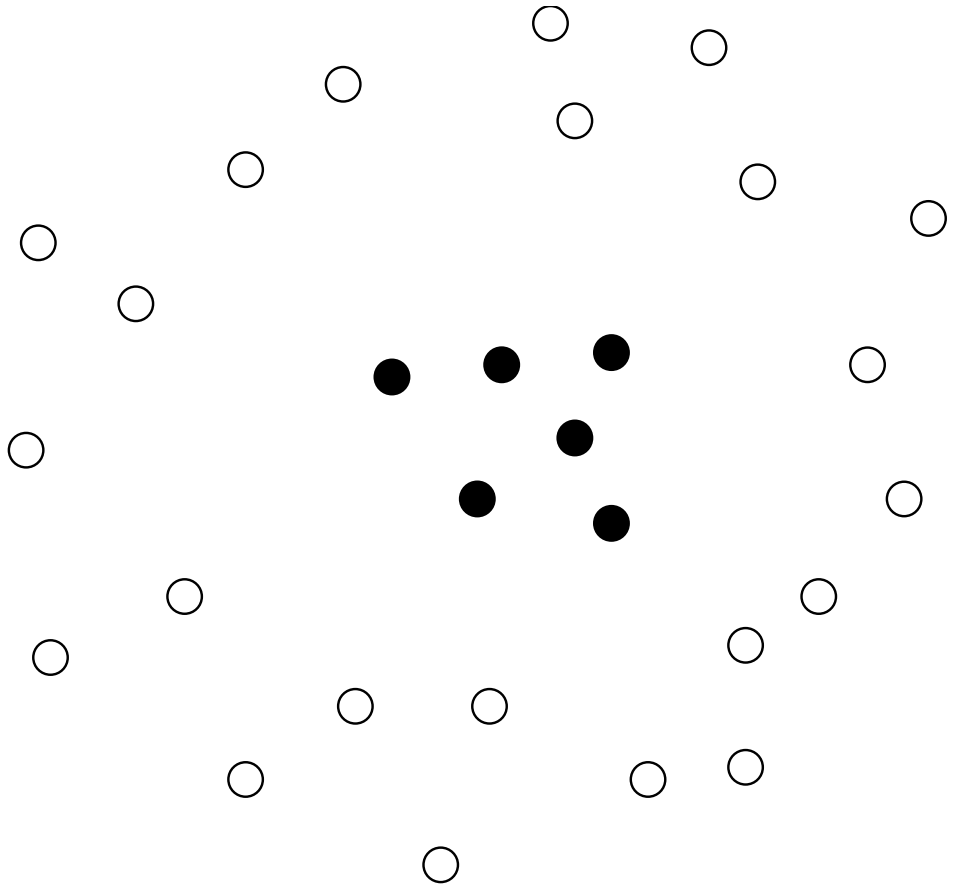
$$\begin{cases} 0 \leq \alpha_i \leq C, & \text{for } i = 1, \ldots, N \\ \sum_{i=1}^{N} \alpha_i y_i = 0. \end{cases}$$

Interpretation: bounded and unbounded support vectors

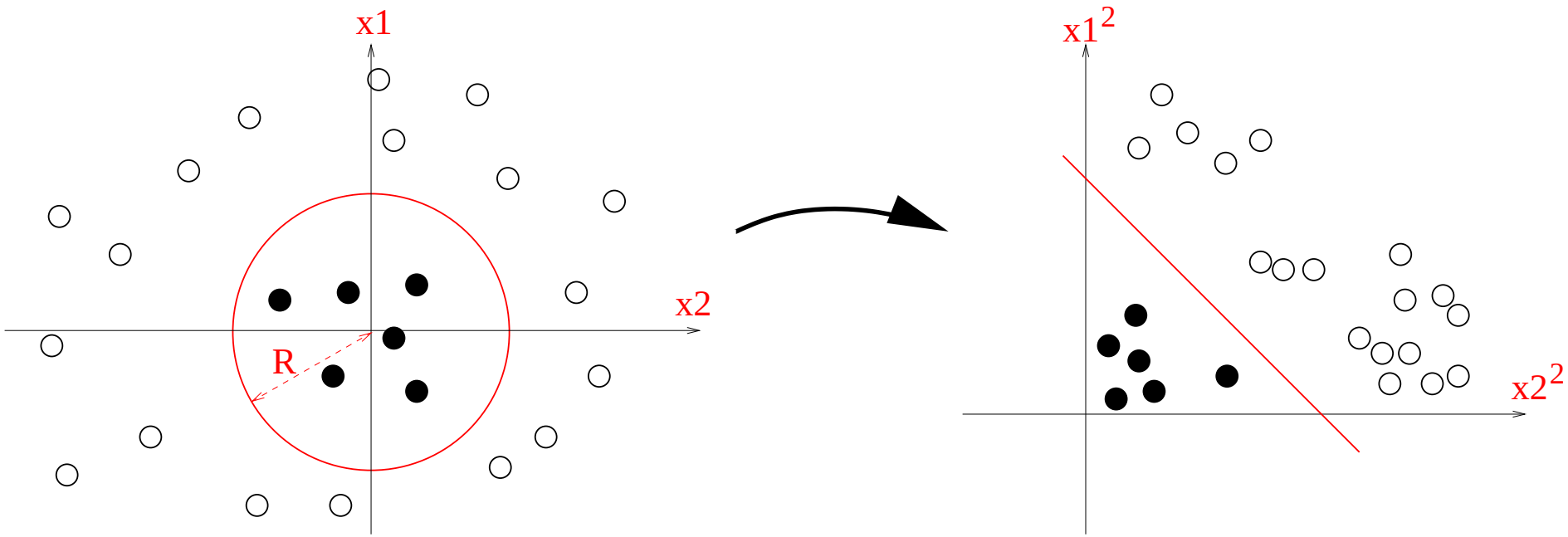


Outline

Sometimes linear classifiers are not interesting



Solution: non-linear mapping to a feature space



Let $\vec{\Phi}(\vec{x}) = (x_1^2, x_2^2)'$, $\vec{w} = (1,1)'$ and $b = 1$. Then the decision function is:

$$f(\vec{x}) = x_1^2 + x_2^2 - R^2 = \vec{w}.\vec{\Phi}(\vec{x}) + b,$$

Kernel (*simple but important*)

For a given mapping Φ from the space of objects $\mathcal{X}$ to some feature space, the kernel of two objects x and x' is the inner product of their images in the features space:

$$\forall x, x' \in \mathcal{X}, \quad K(x, x') = \vec{\Phi}(x).\vec{\Phi}(x').$$

Example: if $\vec{\Phi}(\vec{x}) = (x_1^2, x_2^2)'$, *then*

$$K(\vec{x}, \vec{x'}) = \vec{\Phi}(\vec{x}).\vec{\Phi}(\vec{x'}) = (x_1)^2(x_1')^2 + (x_2)^2(x_2')^2.$$

Training a SVM in the feature space

Replace each $\vec{x}.\vec{x}'$ in the SVM algorithm by $\vec{\Phi}(x).\vec{\Phi}(x') = K(x, x')$
The dual problem is to maximize

$$L(\vec{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j),$$

under the constraints:

$$\begin{cases} 0 \leq \alpha_i \leq C, & \text{for } i = 1, \ldots, N \\ \sum_{i=1}^{N} \alpha_i y_i = 0. \end{cases}$$

Predicting with a SVM in the feature space

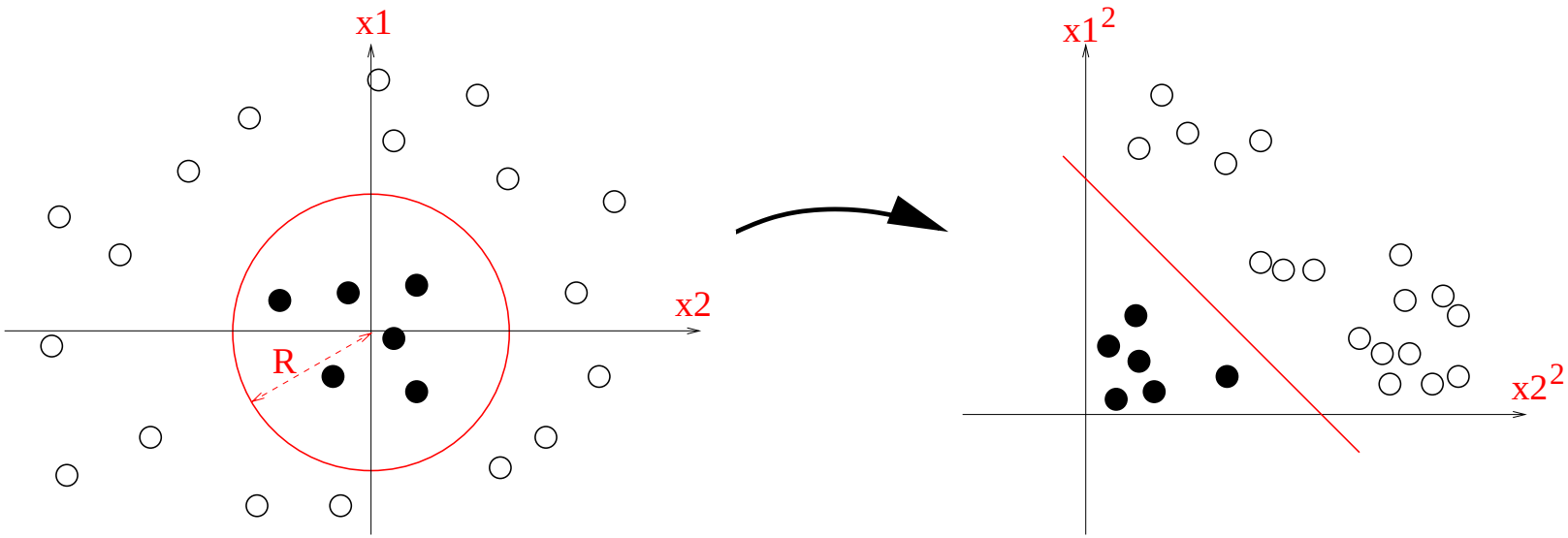
The decision function becomes:

$$\begin{aligned} f(x) &= \vec{w}^* . \vec{\Phi}(x) + b^* \\ &= \sum_{i=1}^{N} \alpha_i K(x_i, x) + b^* . \end{aligned} \tag{2}$$

The kernel trick

- The explicit computation of $\vec{\Phi}(x)$ is not necessary. The kernel $K(x, x')$ is enough. SVM work implicitly in the feature space.
- It is sometimes possible to easily compute kernels which correspond to complex large-dimensional feature spaces.

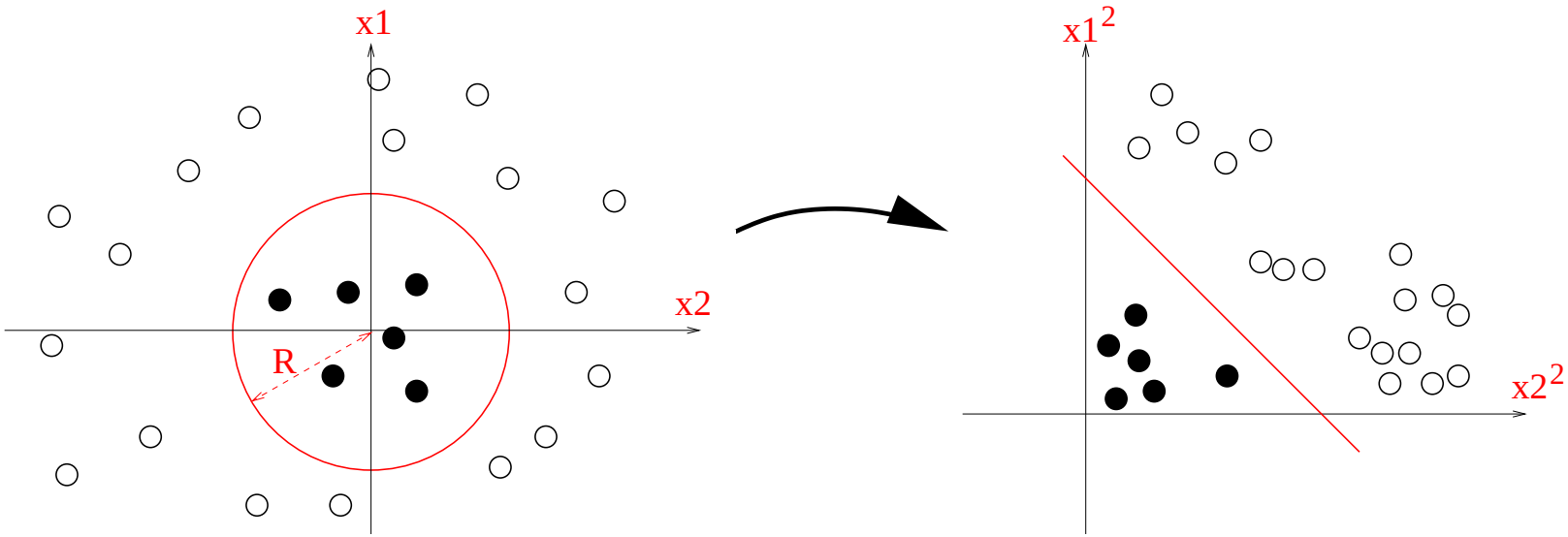
Kernel example: polynomial kernel



For $\vec{x} = (x_1, x_2)^\top \in \mathbb{R}^2$, let $\vec{\Phi}(\vec{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$:

$$\begin{aligned} K(\vec{x}, \vec{x}') &= x_1^2 x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2 x_2'^2 \\ &= \left(x_1x_1' + x_2x_2'\right)^2 \\ &= \left(\vec{x}.\vec{x}'\right)^2 \ . \end{aligned}$$

Kernel example: polynomial kernel



More generally,

$$K(\vec{x}, \vec{x}') = \left(\vec{x}.\vec{x}' + 1\right)^d$$

is an inner product in a feature space of all monomials of degree up to d *(left as exercice.)*

Which functions $K(x, x')$ are kernels?

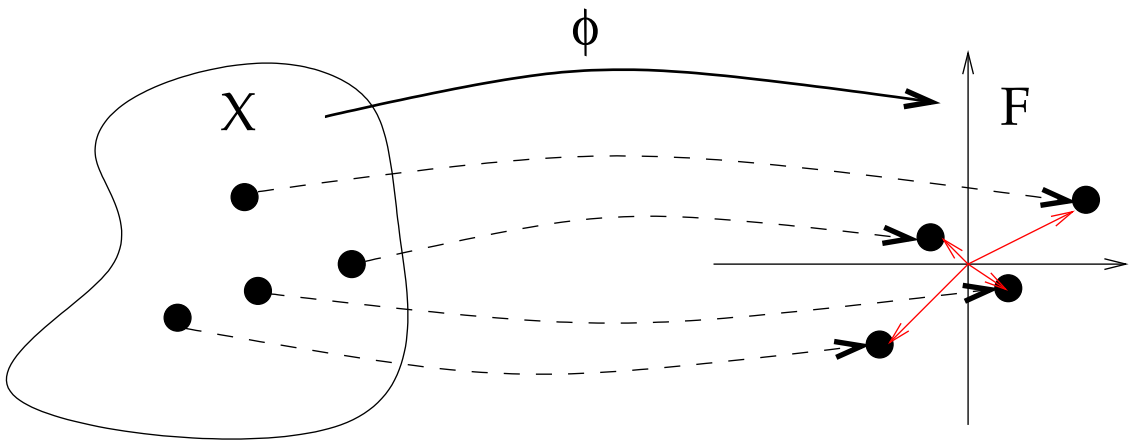
Definition

A function $K(x, x')$ defined on a set $\mathcal{X}$ is a kernel if and only if there exists a features space (Hilbert space) $\mathcal{H}$ and a mapping

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

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

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



Positive Definite (p.d.) functions

Definition

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

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

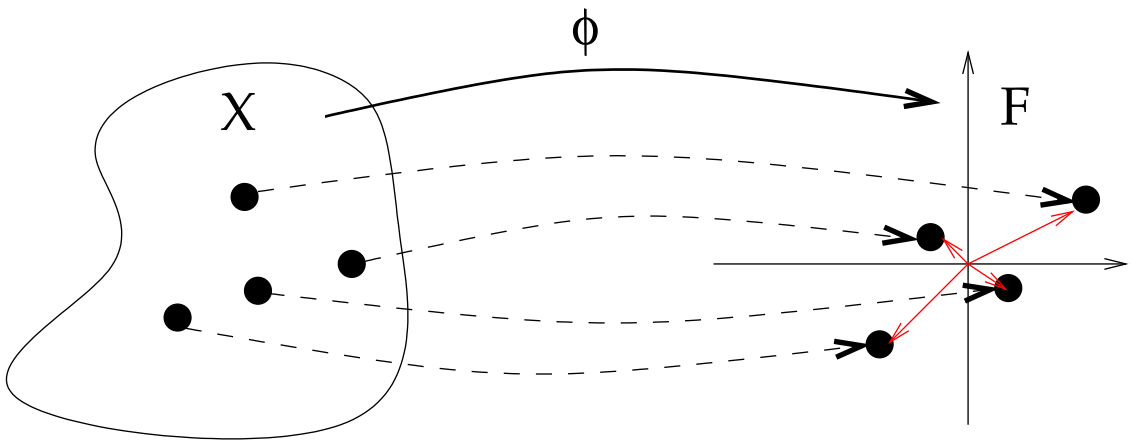
and which satisfies, for all $N \in \mathbb{N}$, $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathcal{X}^N$ et $(a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$:

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

Kernels are p.d. functions

Theorem (Aronszajn, 1950)

K is a kernel if and only if it is a positive definite function.



Proof?

- Kernel $\Longrightarrow$ p.d. function:
  - $\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathbb{R}^d} = \langle \Phi(\mathbf{x}'), \Phi(\mathbf{x})_{\mathbb{R}^d} \rangle$,
  - $\sum_{i=1}^N \sum_{j=1}^N a_i a_j \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle_{\mathbb{R}^d} = \| \sum_{i=1}^N a_i \Phi(\mathbf{x}_i) \|_{\mathbb{R}^d}^2 \geq 0$.
- P.d. function $\Longrightarrow$ kernel: more difficult...

Kernel examples

- Polynomial (on $\mathbb{R}^d$):

$$K(x, x') = (x.x' + 1)^d$$

- Gaussian radial basis function (RBF) (on $\mathbb{R}^d$)

$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)$$

- Laplace kernel (on $\mathbb{R}$)

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

- Min kernel (on $\mathbb{R}_+$)

$$K(x, x') = \min(x, x')$$

Exercice: for each kernel, find a Hilbert space $\mathcal{H}$ and a mapping $\Phi : \mathcal{X} \to \mathcal{H}$ such that $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$

Example: SVM with a Gaussian kernel

- Training:

$$\min_{\alpha \in \mathbb{R}^N} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \exp\left(-\frac{||\vec{x}_i - \vec{x}_j||^2}{2\sigma^2}\right)$$

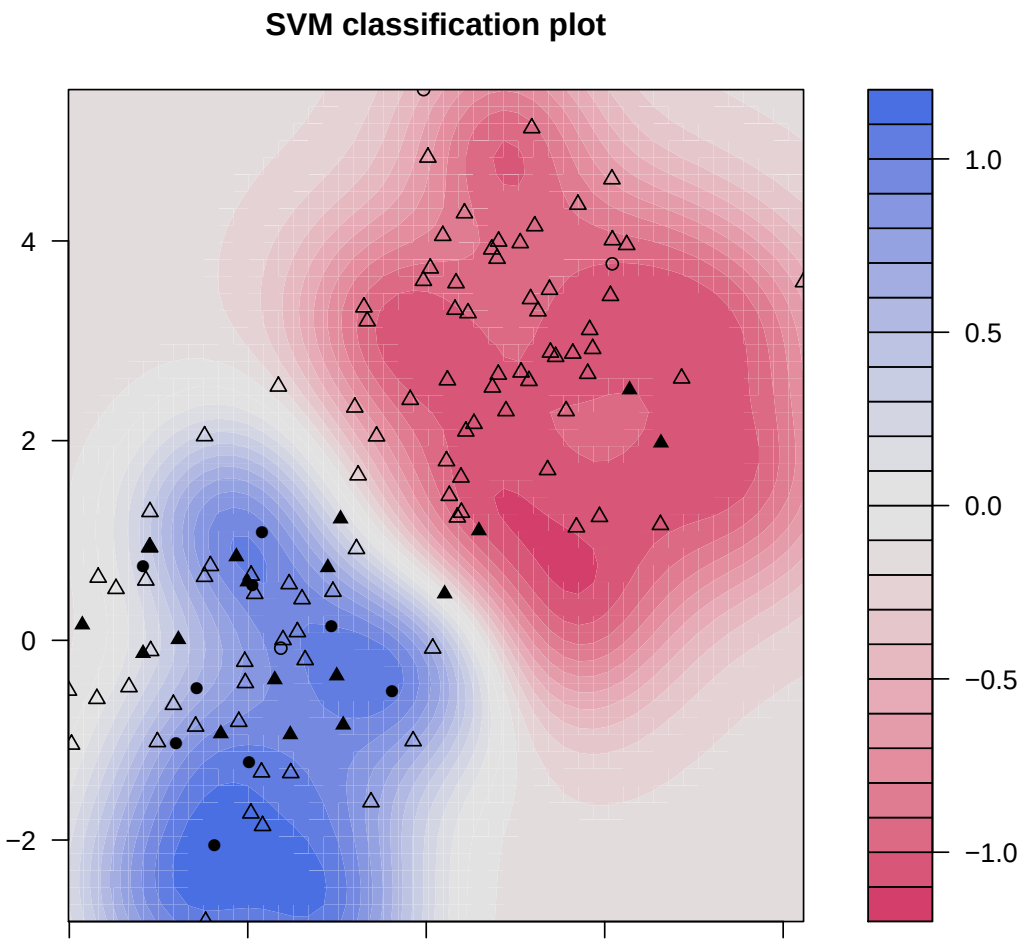
$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad \text{and } \sum_{i=1}^N \alpha_i y_i = 0.$$

- Prediction

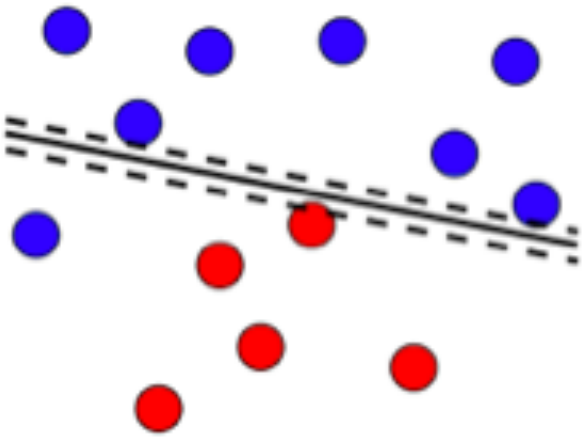
$$f(\vec{x}) = \sum_{i=1}^N \alpha_i \exp\left(-\frac{||\vec{x} - \vec{x}_i||^2}{2\sigma^2}\right)$$

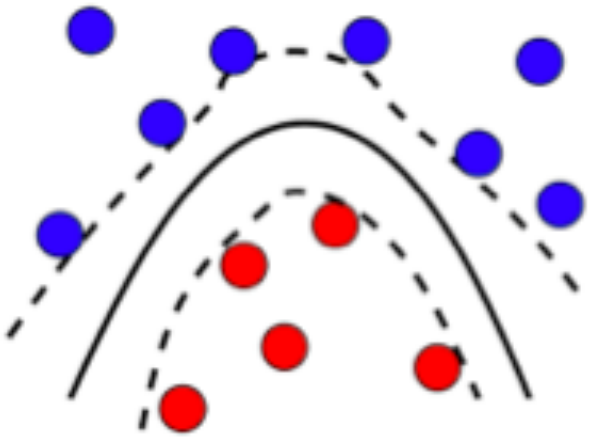
Example: SVM with a Gaussian kernel

$$f(\vec{x}) = \sum_{i=1}^{N} \alpha_i \exp\left(-\frac{||\vec{x} - \vec{x}_i||^2}{2\sigma^2}\right)$$

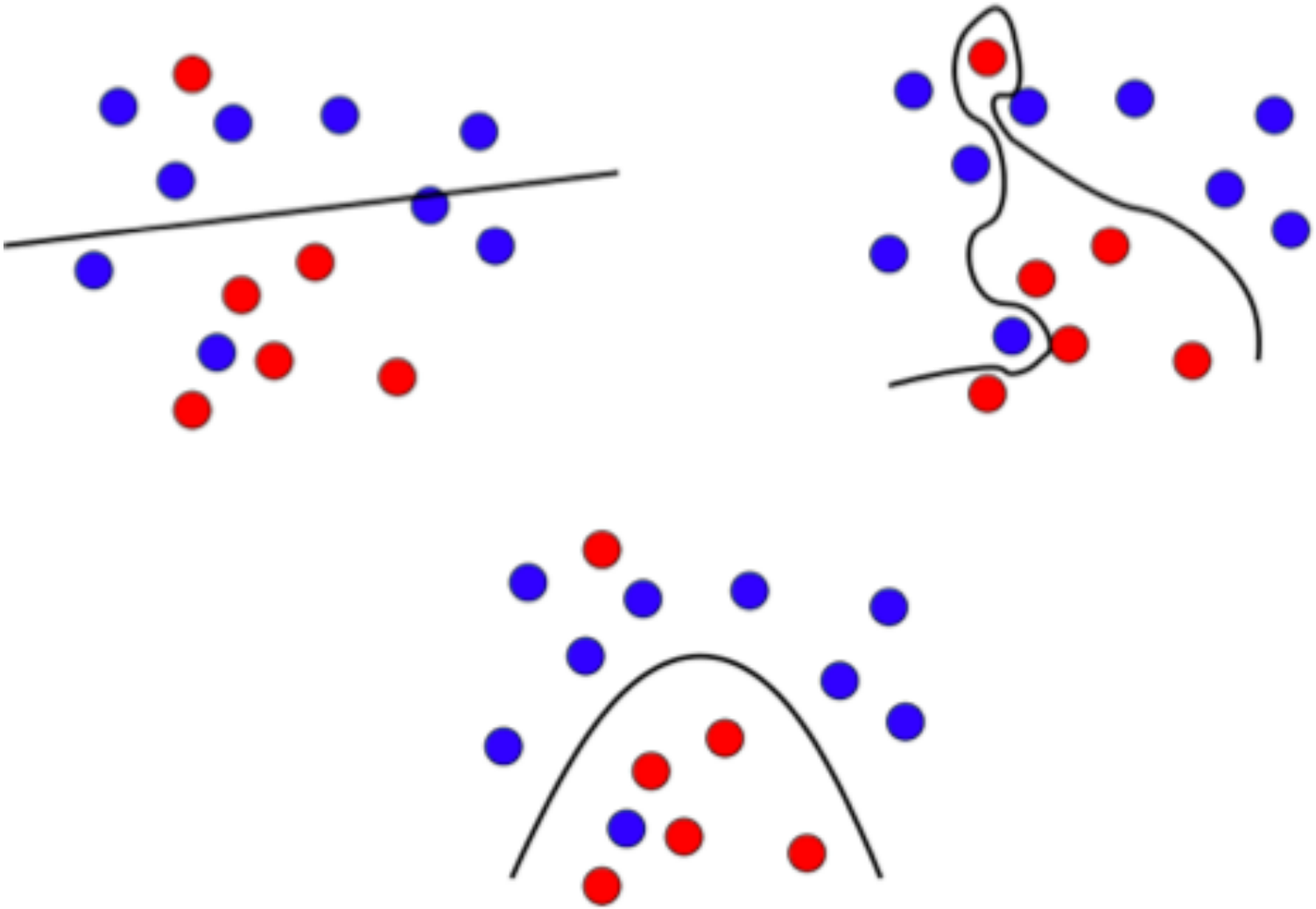


Linear vs nonlinear SVM





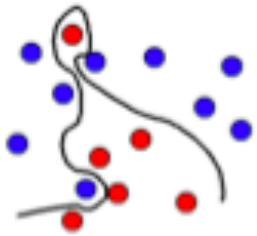
Regularity vs data fitting trade-off



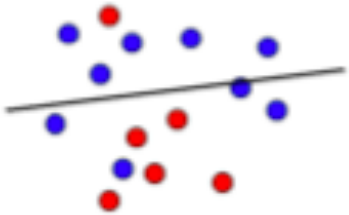
C controls the trade-off

$$\min_f \left\{ \frac{1}{margin(f)} + C \times errors(f) \right\}$$

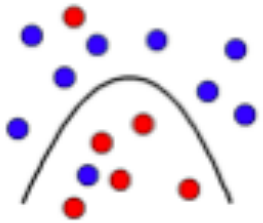
- Large C :
  - makes few errors



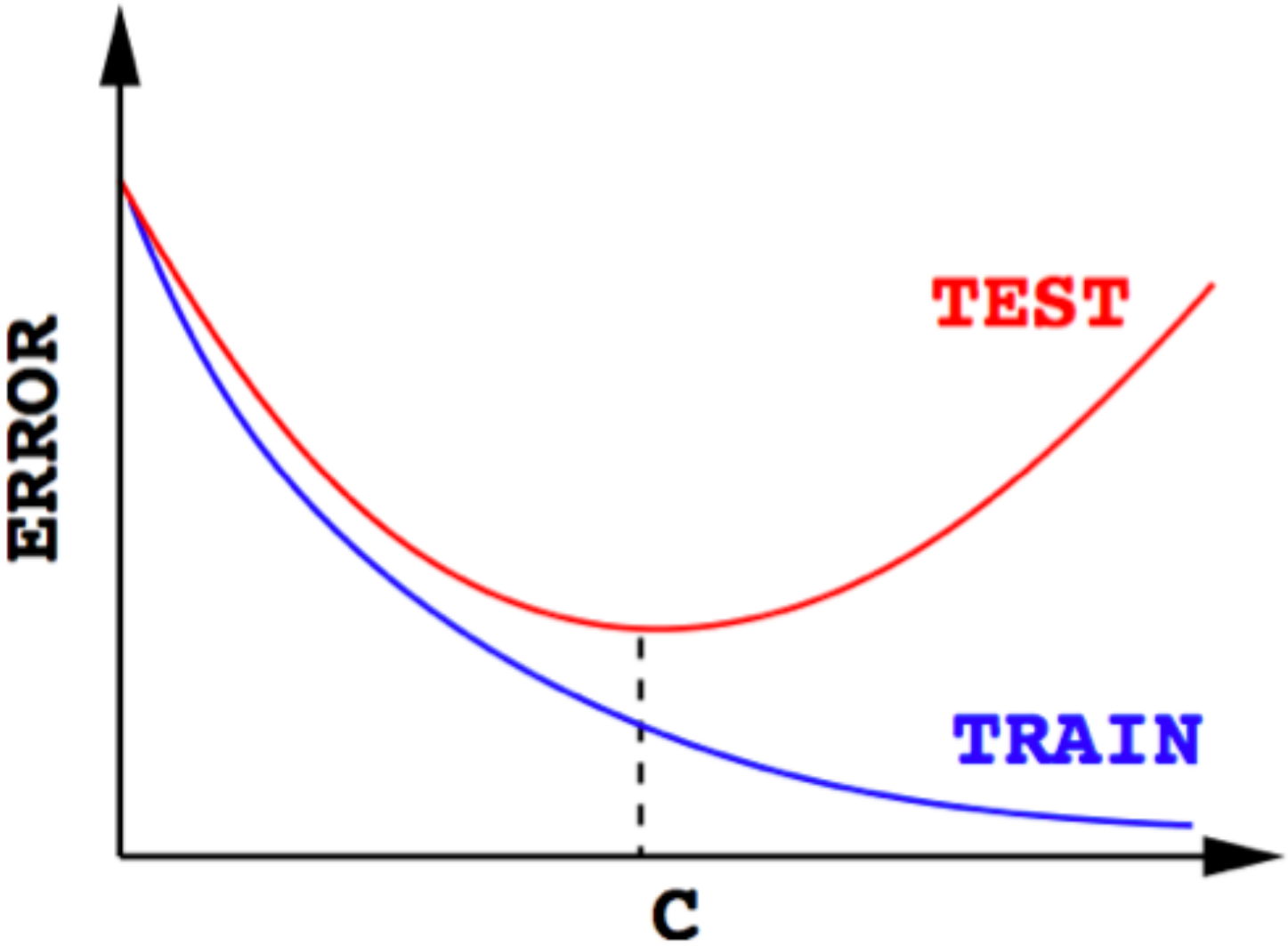
- Small C :
  - ensure a large margin



- Intermediate C:
  - finds a trade-off



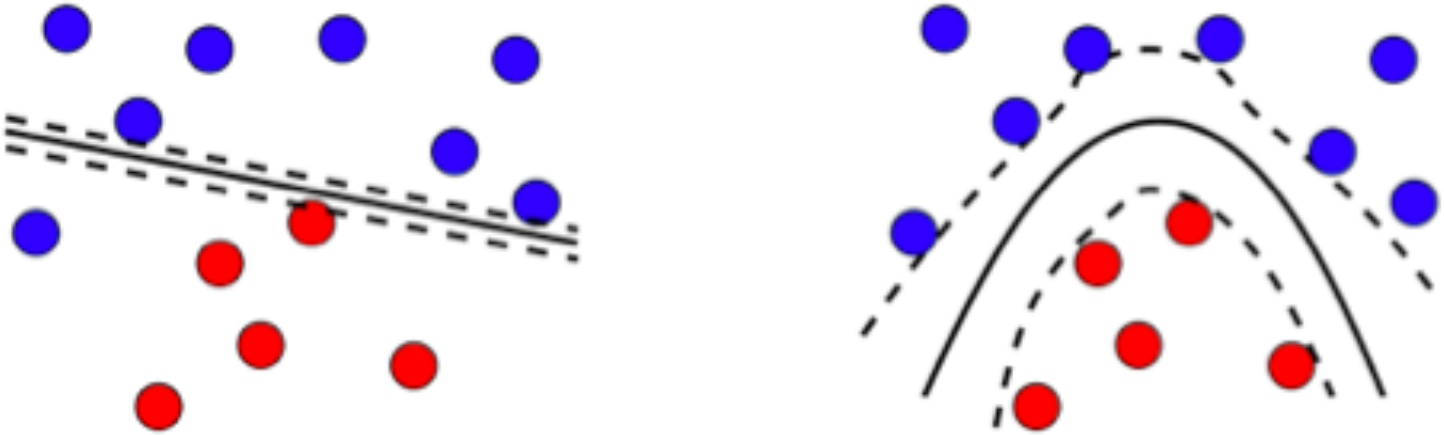
Why it is important to control the trade-off



How to choose C in practice

- Split your dataset in two ("train" and "test")
- Train SVM with different C on the "train" set
- Compute the accuracy of the SVM on the "test" set
- Choose the C which minimizes the "test" error
- (you may repeat this several times = cross-validation)

SVM summary



- Large margin
- Linear or nonlinear (with the kernel trick)
- Control of the regularization / data fitting trade-off with C

Outline

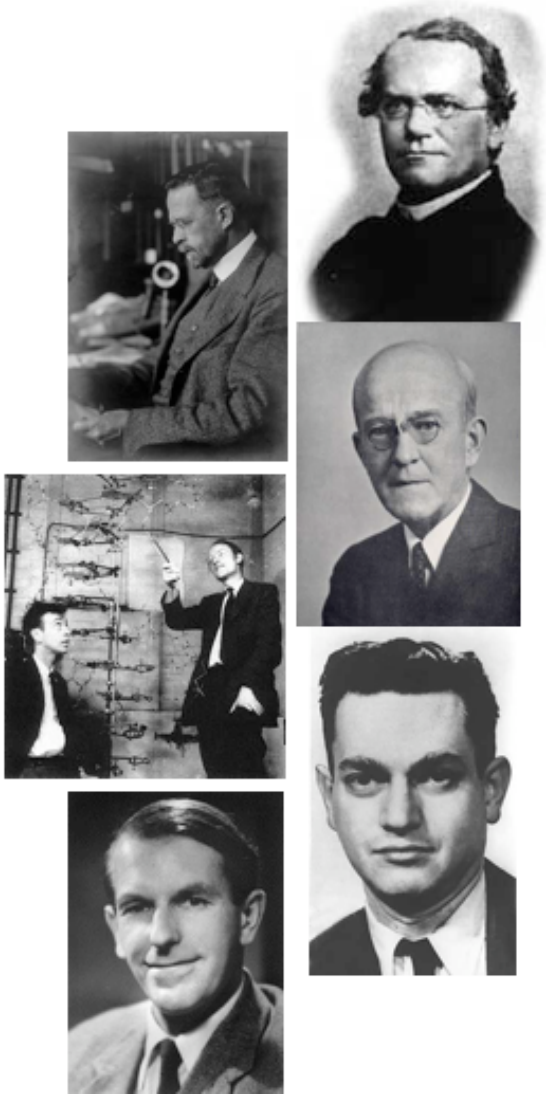
1. SVM and kernel methods
2. Kernels for biological sequences
   - Motivations
   - Feature space approach
   - Using generative models
   - Derive from a similarity measure
   - Application: remote homology detection
3. Kernels for graphs
4. Reconstruction of regulatory networks
5. Supervised graph inference

Part 5

Kernels for Biological Sequences

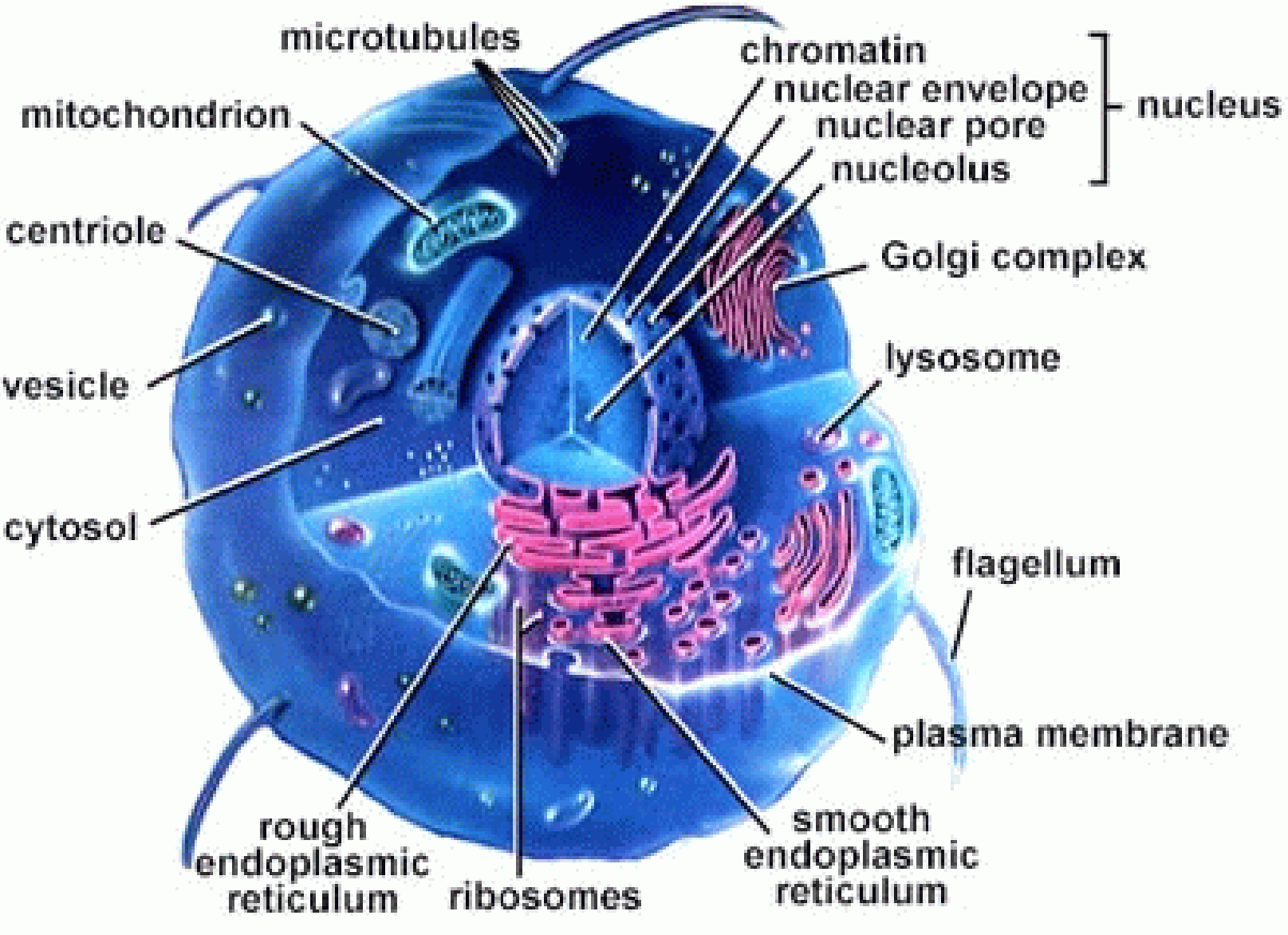
Outline

Short history of genomics

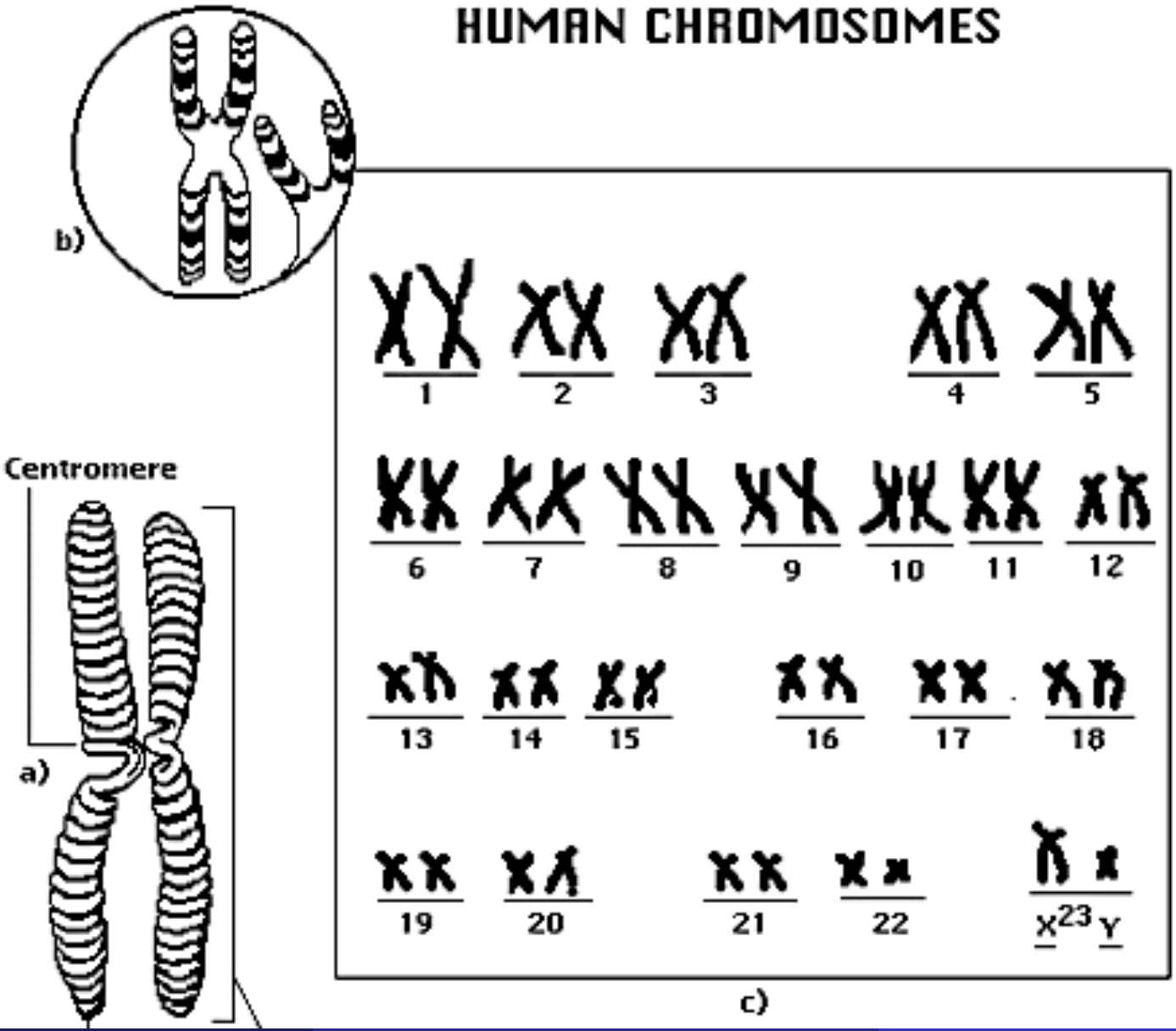


1866 : Laws of heredity (Mendel)
1909 : Morgan and the drosophilists
1944 : DNA supports heredity (Avery)
1953 : Structure of DNA (Crick and Watson)
1966 : Genetic code (Nirenberg)
1960-70 : Genetic engineering
1977 : Method for sequencing (Sanger)
1982 : Creation of Genbank
1990 : Human genome project launched
2003 : Human genome project completed

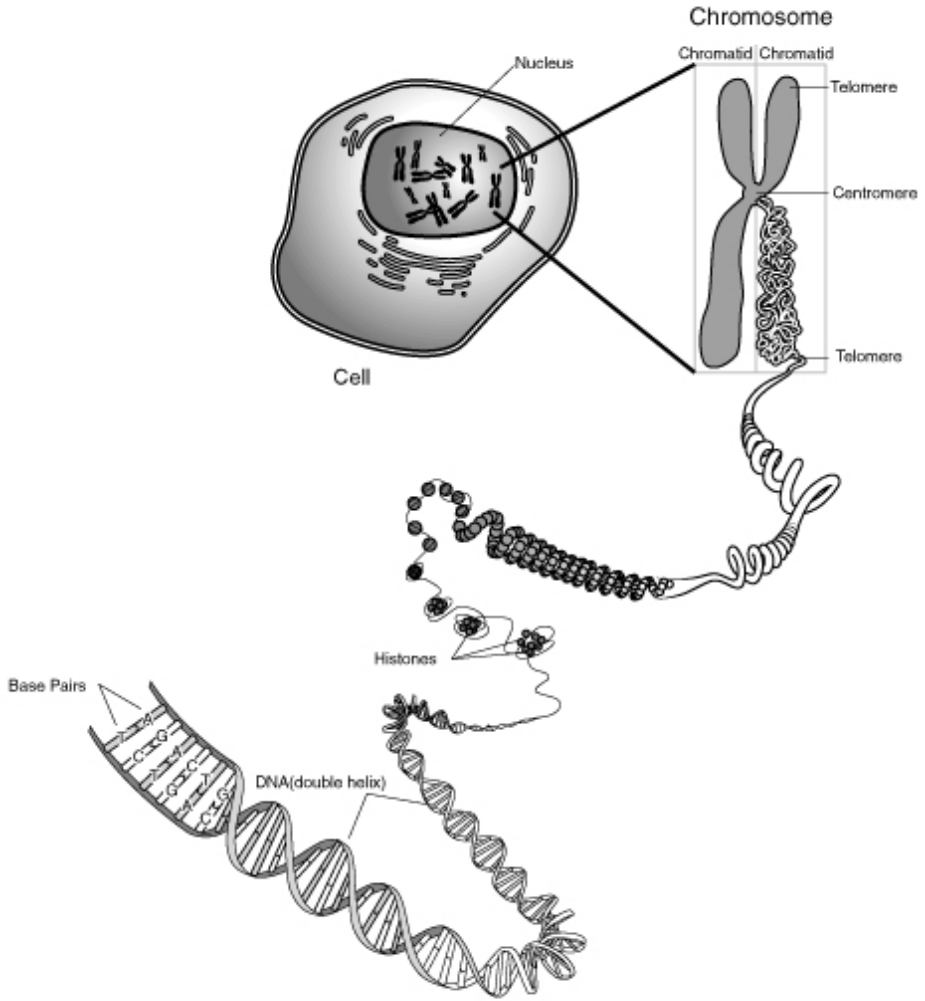
A cell


microtubules
mitochondrion
centriole
vesicle
cytosol
chromatin
nuclear envelope
nuclear pore
nucleolus
nucleus
Golgi complex
lysosome
flagellum
plasma membrane
rough endoplasmic reticulum
ribosomes
smooth endoplasmic reticulum

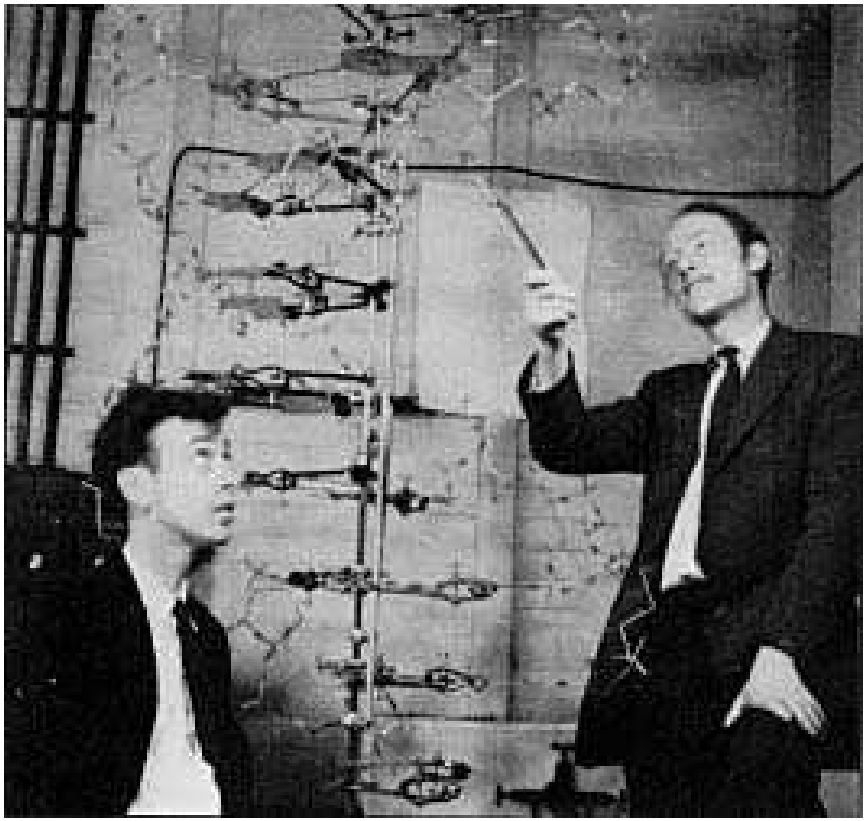
Chromosomes



Chromosomes and DNA

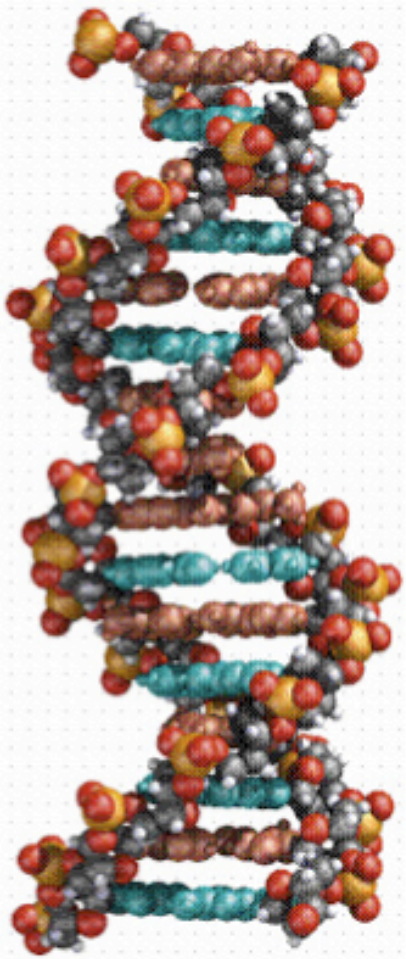


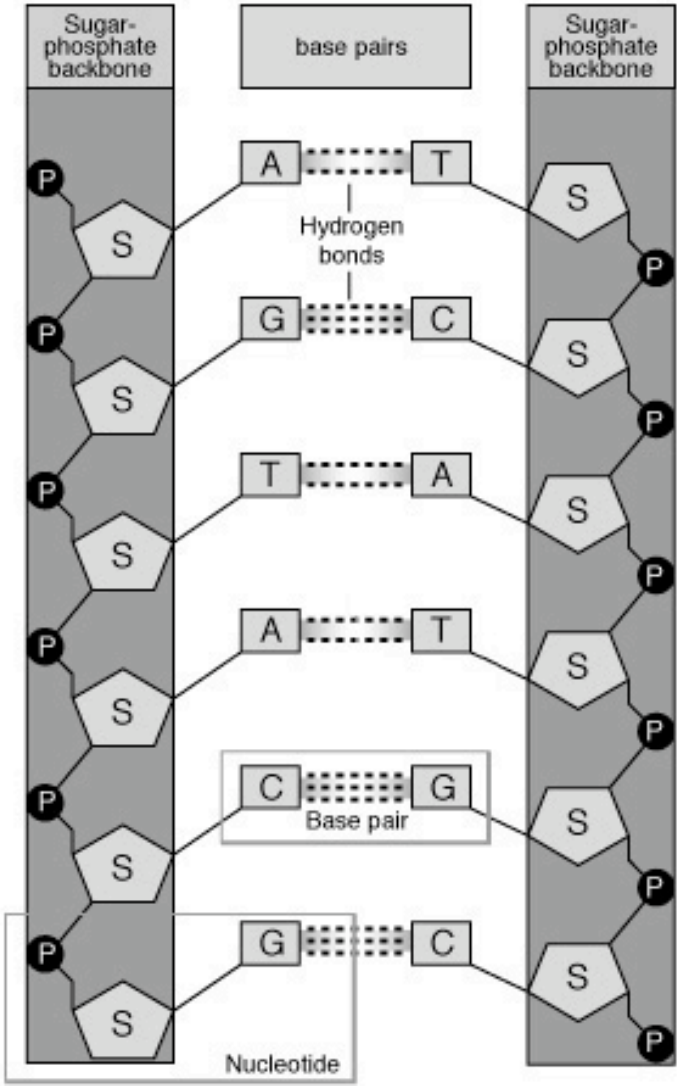
Structure of DNA



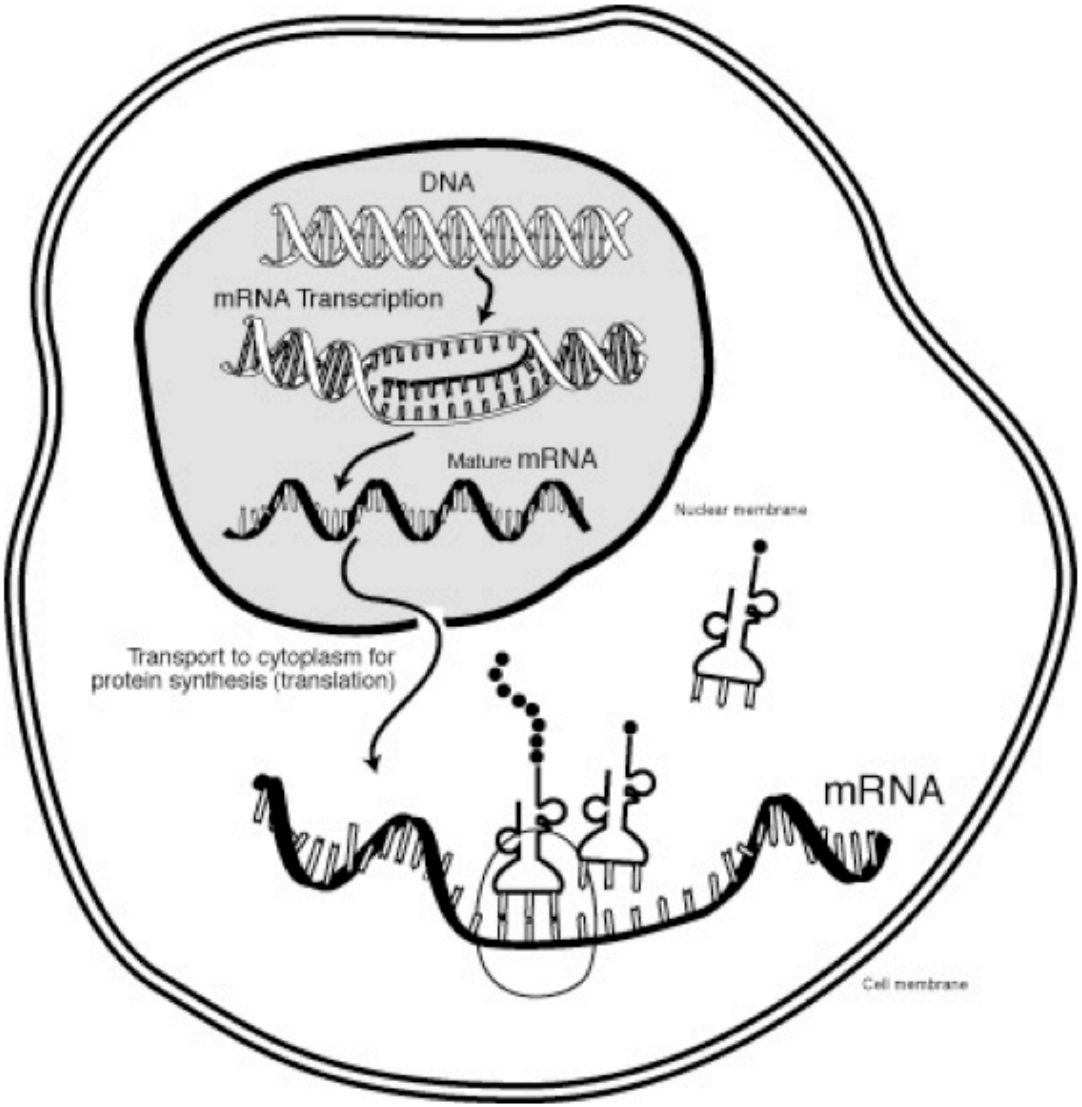
"We wish to suggest a structure for the salt of desoxyribose nucleic acid (D.N.A.). This structure have novel features which are of considerable biological interest" (Watson and Crick, 1953)

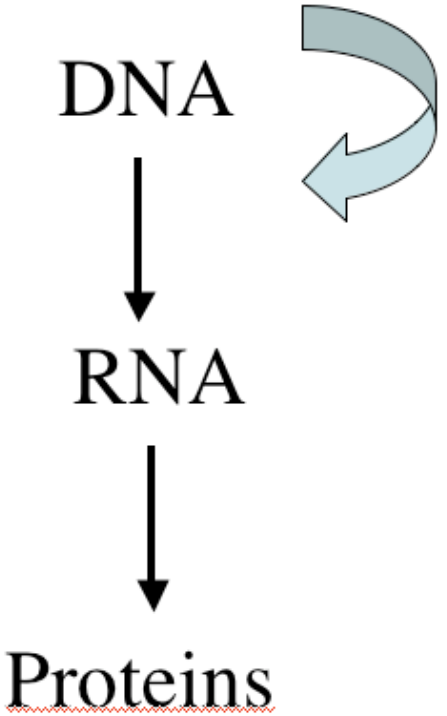
The double helix



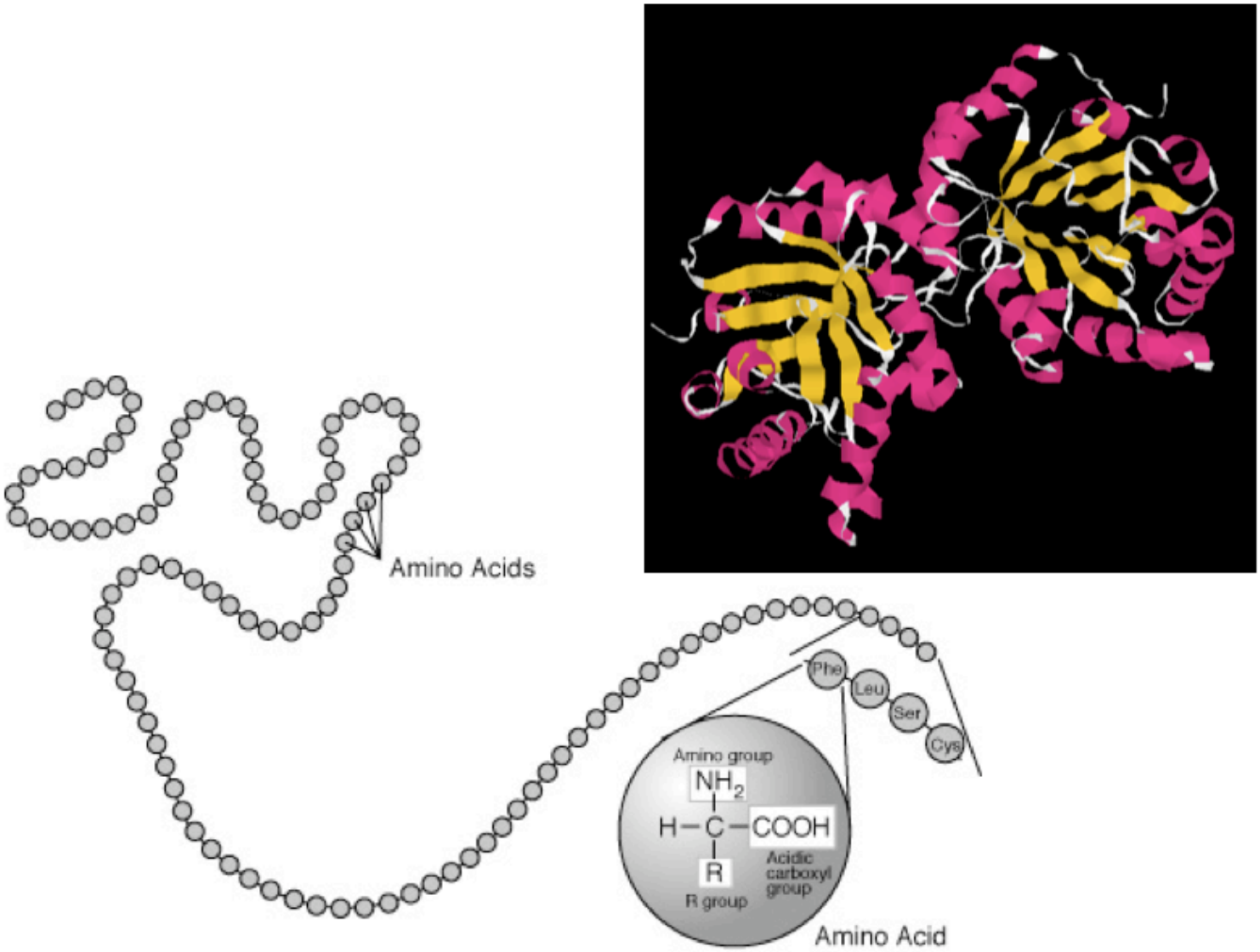


Central dogma

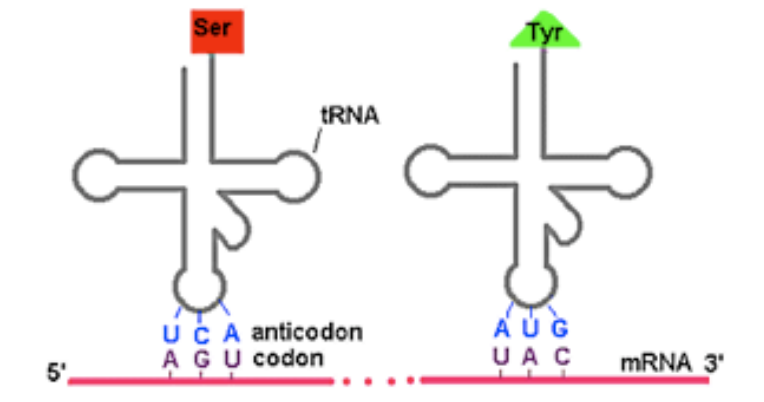




Proteins


Amino Acids
Phe
Leu
Ser
Cys
Amino group
NH_2
H—C—COOH
R
R group
Acidic carboxyl group
Amino Acid

Genetic code



2nd base in codon

1st base in codon	U	C	A	G	3rd base in codon
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	STOP	STOP	A
	Leu	Ser	STOP	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

The Genetic Code

DNA = 4 letters (ATCG)

↓

RNA = 4 letters (AUCG)

↓

Protein = 20 letters (amino acids)

1 amino acid

=

3 nucleotides

Human genome project

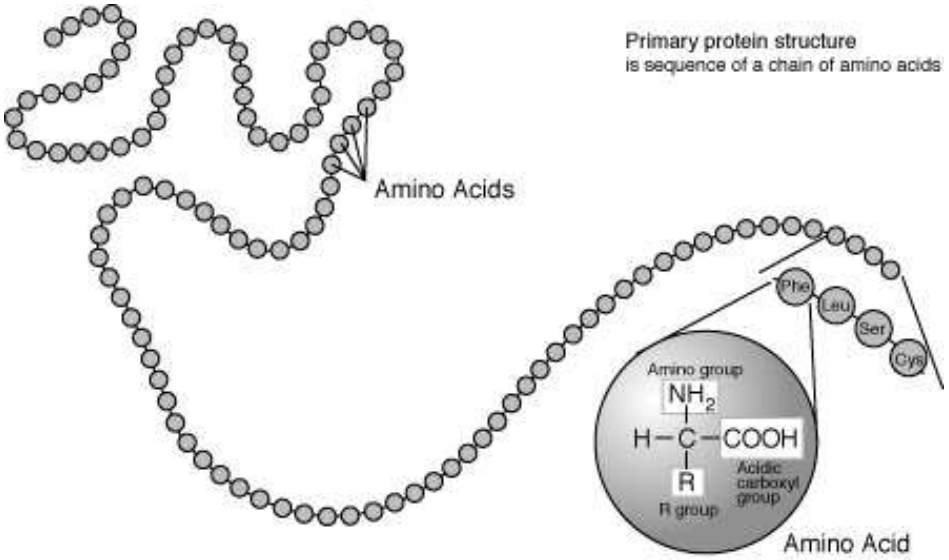
- Goal : sequence the 3,000,000,000 bases of the human genome
- Consortium with 20 labs, 6 countries
- Cost : about 3,000,000,000 USD

2003: End of genomics era

Findings

- About 25,000 genes only (representing 1.2% of the genome)
- Automatic gene finding with graphical models
- 97% of the genome is considered “junk DNA”
- Superposition of a variety of signals (many to be discovered)

Protein sequence



A : Alanine	V : Valine	L : Leucine
F : Phenylalanine	P : Proline	M : Methionine
E : Acide glutamique	K : Lysine	R : Arginine
T : Threonine	C : Cysteine	N : Asparagine
H : Histidine	V : Thyrosine	W : Tryptophane
I : Isoleucine	S : Serine	Q : Glutamine
D : Acide aspartique	G : Glycine	

Challenges with protein sequences

- A protein sequences can be seen as a variable-length sequence over the 20-letter alphabet of amino-acids, e.g., insuline: `FVNQHLCGSHLVEALYLVCGERGFFYTPKA`
- These sequences are produced at a fast rate (result of the sequencing programs)
- Need for algorithms to compare, classify, analyze these sequences
- Applications: classification into functional or structural classes, prediction of cellular localization and interactions, ...

Example: supervised sequence classification

Data (training)

- Secreted proteins:

```
MASKATLLLAFTLLFATCIARHQQRQQQQNQCQLQNIEA...
MARSSLFTFLCLAVFINGCLSQIEQQSPWEFQGSEVW...
MALHTVLIMLSLLPMLEAQNPEHANITIGEPITNETLGWL...
...
```

- Non-secreted proteins:

```
MAPPSVFAEVPQAQPVLVFKLIADFREDPDPRKVNLGVG...
MAHTLGLTQPNSTEPHKISFTAKEIDVIEWKGDILVVG...
MSISESYAKEIKTAFRQFTDFPIEGEQFEDFLPIIGNP..
...
```

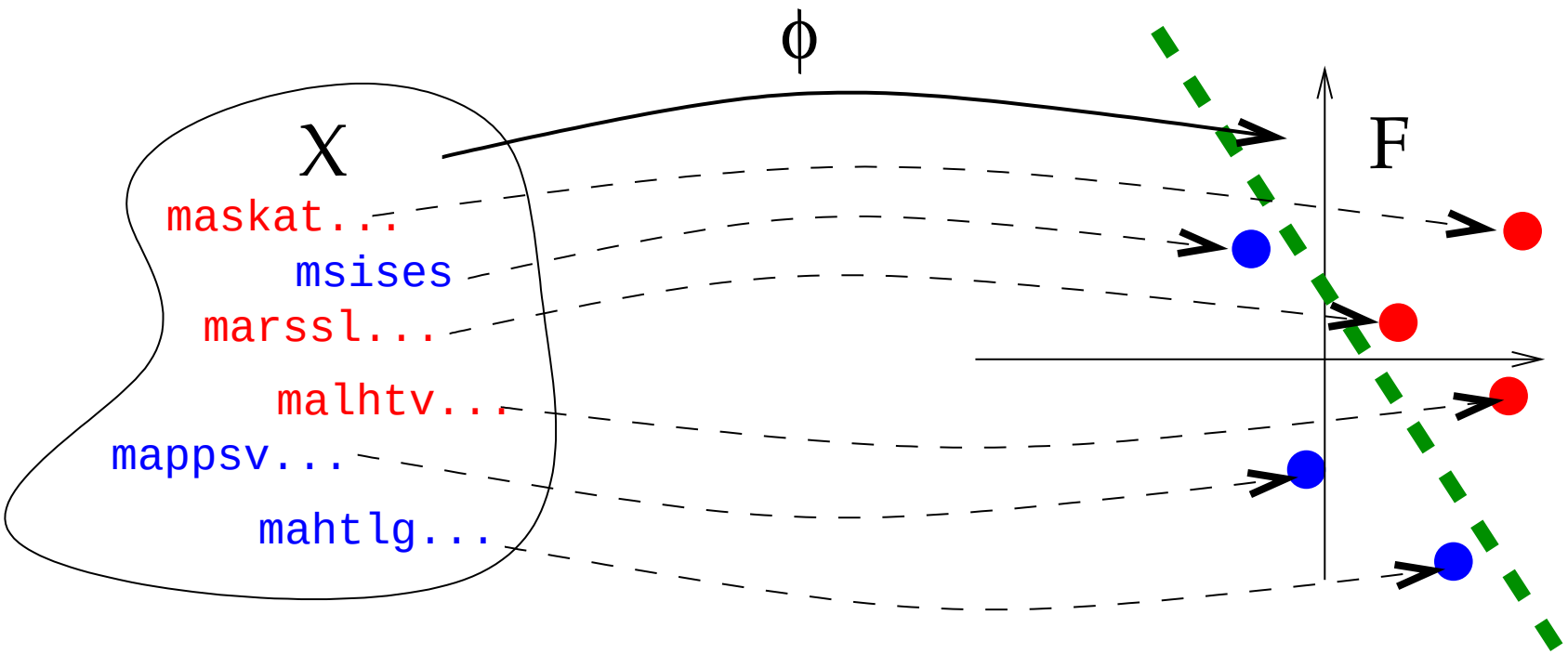
Goal

- Build a classifier to predict whether new proteins are secreted or not.

Supervised classification with vector embedding

The idea

- Map each string $x \in \mathcal{X}$ to a vector $\Phi(x) \in \mathcal{F}$.
- Train a classifier for vectors on the images $\Phi(x_1), \ldots, \Phi(x_n)$ of the training set (nearest neighbor, linear perceptron, logistic regression, support vector machine...)



Kernels for protein sequences

- Kernel methods have been widely investigated since Jaakkola et al.'s seminal paper (1998).
- What is a good kernel?
  - it should be mathematically valid (symmetric, p.d. or c.p.d.)
  - fast to compute
  - adapted to the problem (give good performances)

Kernel engineering for protein sequences

- Define a (possibly high-dimensional) feature space of interest
  - Physico-chemical kernels
  - Spectrum, mismatch, substring kernels
  - Pairwise, motif kernels
- Derive a kernel from a generative model
  - Fisher kernel
  - Mutual information kernel
  - Marginalized kernel
- Derive a kernel from a similarity measure
  - Local alignment kernel

Kernel engineering for protein sequences

- Define a (possibly high-dimensional) feature space of interest
  - Physico-chemical kernels
  - Spectrum, mismatch, substring kernels
  - Pairwise, motif kernels
- Derive a kernel from a generative model
  - Fisher kernel
  - Mutual information kernel
  - Marginalized kernel
- Derive a kernel from a similarity measure
  - Local alignment kernel

Kernel engineering for protein sequences

- Define a (possibly high-dimensional) feature space of interest
  - Physico-chemical kernels
  - Spectrum, mismatch, substring kernels
  - Pairwise, motif kernels
- Derive a kernel from a generative model
  - Fisher kernel
  - Mutual information kernel
  - Marginalized kernel
- Derive a kernel from a similarity measure
  - Local alignment kernel

Outline

Vector embedding for strings

The idea

Represent each sequence $\mathbf{x}$ by a fixed-length numerical vector $\Phi(\mathbf{x}) \in \mathbb{R}^n$. How to perform this embedding?

Physico-chemical kernel

Extract relevant features, such as:

- length of the sequence
- time series analysis of numerical physico-chemical properties of amino-acids along the sequence (e.g., polarity, hydrophobicity), using for example:
  - Fourier transforms (Wang et al., 2004)
  - Autocorrelation functions (Zhang et al., 2003)

$$r_j = \frac{1}{n-j} \sum_{i=1}^{n-j} h_i h_{i+j}$$

Vector embedding for strings

The idea

Represent each sequence $\mathbf{x}$ by a fixed-length numerical vector $\Phi(\mathbf{x}) \in \mathbb{R}^n$. How to perform this embedding?

Physico-chemical kernel

Extract relevant features, such as:

- length of the sequence
- time series analysis of numerical physico-chemical properties of amino-acids along the sequence (e.g., polarity, hydrophobicity), using for example:
  - Fourier transforms (Wang et al., 2004)
  - Autocorrelation functions (Zhang et al., 2003)

$$r_j = \frac{1}{n-j} \sum_{i=1}^{n-j} h_i h_{i+j}$$

Substring indexation

The approach

Alternatively, index the feature space by fixed-length strings, i.e.,

$$\Phi(\mathbf{x}) = (\Phi_u(\mathbf{x}))_{u \in \mathcal{A}^k}$$

where $\Phi_u(\mathbf{x})$ can be:

- the number of occurrences of u in $\mathbf{x}$ (without gaps) : spectrum kernel (Leslie et al., 2002)
- the number of occurrences of u in $\mathbf{x}$ up to m mismatches (without gaps) : mismatch kernel (Leslie et al., 2004)
- the number of occurrences of u in $\mathbf{x}$ allowing gaps, with a weight decaying exponentially with the number of gaps : substring kernel (Lohdi et al., 2002)

Example: spectrum kernel (1/2)

Kernel definition

- The 3-spectrum of

$$\mathbf{x} = \texttt{CGGSLIAMMWFGV}$$

is:

$$\texttt{(CGG,GGS,GSL,SLI,LIA,IAM,AMM,MMW,MWF,WFG,FGV)} \ .$$

- Let $\Phi_u(\mathbf{x})$ denote the number of occurrences of u in $\mathbf{x}$. The k-spectrum kernel is:

$$K(\mathbf{x}, \mathbf{x}') := \sum_{u \in \mathcal{A}^k} \Phi_u(\mathbf{x}) \Phi_u(\mathbf{x}') \ .$$

Example: spectrum kernel (2/2)

Implementation

- The computation of the kernel is formally a sum over $|\mathcal{A}|^k$ terms, but at most $|\mathbf{x}| - k + 1$ terms are non-zero in $\Phi(\mathbf{x}) \implies$ Computation in $O(|\mathbf{x}| + |\mathbf{x}'|)$ with pre-indexation of the strings.
- Fast classification of a sequence $\mathbf{x}$ in $O(|\mathbf{x}|)$:

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) = \sum_u w_u \Phi_u(\mathbf{x}) = \sum_{i=1}^{|\mathbf{x}|-k+1} w_{x_i \ldots x_{i+k-1}}.$$

Remarks

- Work with any string (natural language, time series...)
- Fast and scalable, a good default method for string classification.
- Variants allow matching of k-mers up to m mismatches.

Example 2: Substring kernel (1/11)

Definition

- For $1 \leq k \leq n \in \mathbb{N}$, we denote by $\mathcal{I}(k, n)$ the set of sequences of indices $\mathbf{i} = (i_1, \ldots, i_k)$, with $1 \leq i_1 < i_2 < \ldots < i_k \leq n$.
- For a string $\mathbf{x} = x_1 \ldots x_n \in \mathcal{X}$ of length n, for a sequence of indices $\mathbf{i} \in \mathcal{I}(k, n)$, we define a substring as:

$$\mathbf{x}(\mathbf{i}) := x_{i_1} x_{i_2} \ldots x_{i_k} .$$

- The length of the substring is:

$$l(\mathbf{i}) = i_k - i_1 + 1.$$

Example 2: Substring kernel (2/11)

Example

ABRACADABRA

- $\mathbf{i} = (3, 4, 7, 8, 10)$
- $\mathbf{x}(\mathbf{i}) =$RADAR
- $l(\mathbf{i}) = 10 - 3 + 1 = 8$

Example 2: Substring kernel (3/11)

The kernel

- Let $k \in \mathbb{N}$ and $\lambda \in \mathbb{R}^+$ fixed. For all $\mathbf{u} \in \mathcal{A}^k$, let $\Phi_{\mathbf{u}} : \mathcal{X} \to \mathbb{R}$ be defined by:

$$\forall \mathbf{x} \in \mathcal{X}, \quad \Phi_{\mathbf{u}}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathcal{I}(k, |\mathbf{x}|):\ \mathbf{x}(\mathbf{i}) = \mathbf{u}} \lambda^{l(\mathbf{i})} .$$

- The substring kernel is the p.d. kernel defined by:

$$\forall (\mathbf{x}, \mathbf{x}') \in \mathcal{X}^2, \quad K_{k,\lambda}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \mathcal{A}^k} \Phi_{\mathbf{u}}(\mathbf{x}) \Phi_{\mathbf{u}}(\mathbf{x}') .$$

Example 2: Substring kernel (4/11)

Example

u	ca	ct	at	ba	bt	cr	ar	br
Φ_u(cat)	λ^2	λ^3	λ^2	0	0	0	0	0
Φ_u(car)	λ^2	0	0	0	0	λ^3	λ^2	0
Φ_u(bat)	0	0	λ^2	λ^2	λ^3	0	0	0
Φ_u(bar)	0	0	0	λ^2	0	0	λ^2	λ^3

$$\begin{cases} K(\text{cat,cat}) = K(\text{car,car}) = 2\lambda^4 + \lambda^6 \\ K(\text{cat,car}) = \lambda^4 \\ K(\text{cat,bar}) = 0 \end{cases}$$

Example 2: Substring kernel (5/11)

Kernel computation

- We need to compute, for any pair $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, the kernel:

$$K_{n,\lambda}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \mathcal{A}^k} \Phi_{\mathbf{u}}(\mathbf{x}) \Phi_{\mathbf{u}}(\mathbf{x}')$$
$$= \sum_{\mathbf{u} \in \mathcal{A}^k} \sum_{\mathbf{i}:\mathbf{x}(\mathbf{i})=\mathbf{u}} \sum_{\mathbf{i}':\mathbf{x}'(\mathbf{i}')=\mathbf{u}} \lambda^{l(\mathbf{i})+l(\mathbf{i}')} .$$

- Enumerating the substrings is too slow (of order $|\mathbf{x}|^k$).

Example 2: Substring kernel (6/11)

Kernel computation (cont.)

- For $\mathbf{u} \in \mathcal{A}^k$ remember that:

$$\Phi_{\mathbf{u}}(\mathbf{x}) = \sum_{\mathbf{i}:\mathbf{x}(\mathbf{i})=\mathbf{u}} \lambda^{i_n - i_1 + 1} .$$

- Let now:

$$\Psi_{\mathbf{u}}(\mathbf{x}) = \sum_{\mathbf{i}:\mathbf{x}(\mathbf{i})=\mathbf{u}} \lambda^{|\mathbf{x}| - i_1 + 1} .$$

Example 2: Substring kernel (7/11)

Kernel computation (cont.)

Let us note $\mathbf{x}(1,j) = x_1 \dots x_j$. A simple rewriting shows that, if we note $a \in \mathcal{A}$ the last letter of $\mathbf{u}$ ($\mathbf{u} = \mathbf{v}a$):

$$\Phi_{\mathbf{v}a}(\mathbf{x}) = \sum_{j \in [1,|\mathbf{x}|]: x_j = a} \Psi_{\mathbf{v}}(\mathbf{x}(1,j-1))\,\lambda\,,$$

and

$$\Psi_{\mathbf{v}a}(\mathbf{x}) = \sum_{j \in [1,|\mathbf{x}|]: x_j = a} \Psi_{\mathbf{v}}(\mathbf{x}(1,j-1))\,\lambda^{|\mathbf{x}|-j+1}\,.$$

Example 2: Substring kernel (8/11)

Kernel computation (cont.)

Moreover we observe that if the string is of the form $\mathbf{x}a$ (i.e., the last letter is $a \in \mathcal{A}$), then:

- If the last letter of $\mathbf{u}$ is not a:

$$\begin{cases} \Phi_{\mathbf{u}}(\mathbf{x}a) & = \Phi_{\mathbf{u}}(\mathbf{x}) \,, \\ \Psi_{\mathbf{u}}(\mathbf{x}a) & = \lambda \Psi_{\mathbf{u}}(\mathbf{x}) \,. \end{cases}$$

- If the last letter of $\mathbf{u}$ is a (i.e., $\mathbf{u} = \mathbf{v}a$ with $\mathbf{v} \in \mathcal{A}^{n-1}$):

$$\begin{cases} \Phi_{\mathbf{v}a}(\mathbf{x}a) & = \Phi_{\mathbf{v}a}(\mathbf{x}) + \lambda \Psi_{\mathbf{v}}(\mathbf{x}) \,, \\ \Psi_{\mathbf{v}a}(\mathbf{x}a) & = \lambda \Psi_{\mathbf{v}a}(\mathbf{x}) + \lambda \Psi_{\mathbf{v}}(\mathbf{x}) \,. \end{cases}$$

Example 2: Substring kernel (9/11)

Kernel computation (cont.)

Let us now show how the function:

$$B_n(\mathbf{x}, \mathbf{x}') := \sum_{\mathbf{u} \in \mathcal{A}^n} \Psi_{\mathbf{u}}(\mathbf{x}) \Psi_{\mathbf{u}}(\mathbf{x}')$$

and the kernel:

$$K_n(\mathbf{x}, \mathbf{x}') := \sum_{\mathbf{u} \in \mathcal{A}^n} \Phi_{\mathbf{u}}(\mathbf{x}) \Phi_{\mathbf{u}}(\mathbf{x}')$$

can be computed recursively. We note that:

$$\begin{cases} B_0(\mathbf{x}, \mathbf{x}') = K_0(\mathbf{x}, \mathbf{x}') = 0 & \text{for all } \mathbf{x}, \mathbf{x}' \\ B_k(\mathbf{x}, \mathbf{x}') = K_k(\mathbf{x}, \mathbf{x}') = 0 & \text{if } \min(|\mathbf{x}|, |\mathbf{x}'|) < k \end{cases}$$

Example 2: Substring kernel (10/11)

Recursive computation of B_n

$$
\begin{aligned}
&B_n\left(\mathbf{x}a, \mathbf{x}'\right) \\
&= \sum_{\mathbf{u}\in\mathcal{A}^n} \Psi_{\mathbf{u}}\left(\mathbf{x}a\right)\Psi_{\mathbf{u}}\left(\mathbf{x}'\right) \\
&= \lambda \sum_{\mathbf{u}\in\mathcal{A}^n} \Psi_{\mathbf{u}}\left(\mathbf{x}\right)\Psi_{\mathbf{u}}\left(\mathbf{x}'\right) + \lambda \sum_{\mathbf{v}\in\mathcal{A}^{n-1}} \Psi_{\mathbf{v}}\left(\mathbf{x}\right)\Psi_{\mathbf{v}a}\left(\mathbf{x}'\right) \\
&= \lambda B_n\left(\mathbf{x}, \mathbf{x}'\right) + \\
&\quad \lambda \sum_{\mathbf{v}\in\mathcal{A}^{n-1}} \Psi_{\mathbf{v}}\left(\mathbf{x}\right) \left(\sum_{j\in[1,|\,\mathbf{x}'\,|]:x'_j=a} \Psi_{\mathbf{v}}\left(\mathbf{x}'\left(1,j-1\right)\right)\lambda^{|\,\mathbf{x}'\,|-j+1} \right) \\
&= \lambda B_n\left(\mathbf{x}, \mathbf{x}'\right) + \sum_{j\in[1,|\,\mathbf{x}'\,|]:x'_j=a} B_{n-1}\left(\mathbf{x}, \mathbf{x}'\left(1,j-1\right)\right)\lambda^{|\,\mathbf{x}'\,|-j+2}
\end{aligned}
$$

Example 2: Substring kernel (10/11)

Recursive computation of K_n

$$
\begin{aligned}
&K_n\left(\mathbf{x}a, \mathbf{x}'\right) \\
&= \sum_{\mathbf{u}\in\mathcal{A}^n} \Phi_{\mathbf{u}}\left(\mathbf{x}a\right)\Phi_{\mathbf{u}}\left(\mathbf{x}'\right) \\
&= \sum_{\mathbf{u}\in\mathcal{A}^n} \Phi_{\mathbf{u}}\left(\mathbf{x}\right)\Phi_{\mathbf{u}}\left(\mathbf{x}'\right) + \lambda \sum_{\mathbf{v}\in\mathcal{A}^{n-1}} \Psi_{\mathbf{v}}\left(\mathbf{x}\right)\Phi_{\mathbf{v}a}\left(\mathbf{x}'\right) \\
&= K_n\left(\mathbf{x}, \mathbf{x}'\right) + \\
&\quad \lambda \sum_{\mathbf{v}\in\mathcal{A}^{n-1}} \Psi_{\mathbf{v}}\left(\mathbf{x}\right) \left(\sum_{j\in[1,|\,\mathbf{x}'\,|]:x'_j=a} \Psi_{\mathbf{v}}\left(\mathbf{x}'\left(1,j-1\right)\right)\lambda \right) \\
&= \lambda K_n\left(\mathbf{x}, \mathbf{x}'\right) + \lambda^2 \sum_{j\in[1,|\,\mathbf{x}'\,|]:x'_j=a} B_{n-1}\left(\mathbf{x}, \mathbf{x}'\left(1,j-1\right)\right)
\end{aligned}
$$

Summary: Substring indexation

- Implementation in $O(|\mathbf{x}| + |\mathbf{x}'|)$ in memory and time for the spectrum and mismatch kernels (with suffix trees)
- Implementation in $O(|\mathbf{x}| \times |\mathbf{x}'|)$ in memory and time for the substring kernels
- The feature space has high dimension ($|\mathcal{A}|^k$), so learning requires regularized methods (such as SVM)

Dictionary-based indexation

The approach

- Chose a dictionary of sequences $\mathcal{D} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$
- Chose a measure of similarity $s(\mathbf{x}, \mathbf{x}')$
- Define the mapping $\Phi_{\mathcal{D}}(\mathbf{x}) = (s(\mathbf{x}, \mathbf{x}_i))_{\mathbf{x}_i \in \mathcal{D}}$

Examples

This includes:

- Motif kernels (Logan et al., 2001): the dictionary is a library of motifs, the similarity function is a matching function
- Pairwise kernel (Liao & Noble, 2003): the dictionary is the training set, the similarity is a classical measure of similarity between sequences.

Dictionary-based indexation

The approach

- Chose a dictionary of sequences $\mathcal{D} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$
- Chose a measure of similarity $s(\mathbf{x}, \mathbf{x}')$
- Define the mapping $\Phi_{\mathcal{D}}(\mathbf{x}) = (s(\mathbf{x}, \mathbf{x}_i))_{\mathbf{x}_i \in \mathcal{D}}$

Examples

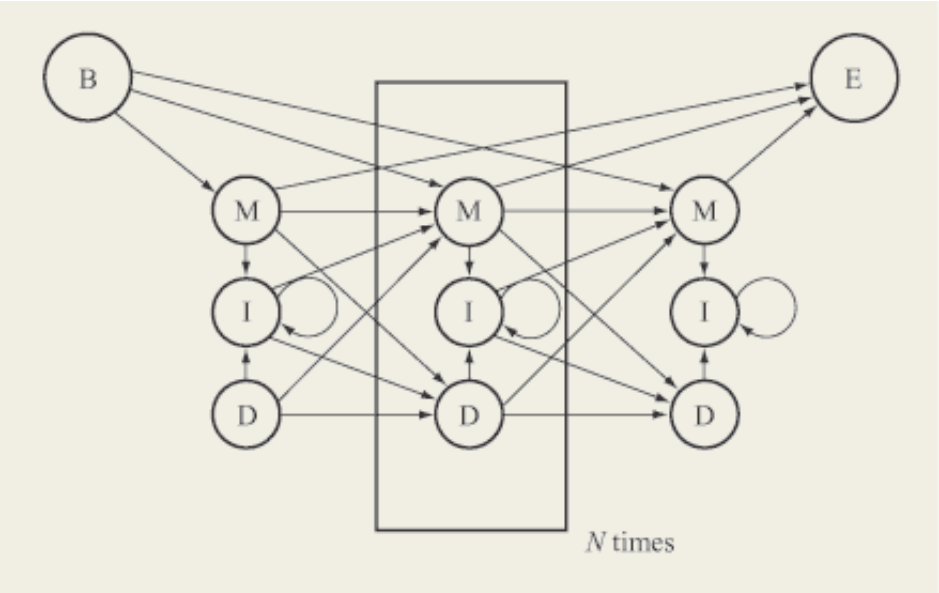
This includes:

- Motif kernels (Logan et al., 2001): the dictionary is a library of motifs, the similarity function is a matching function
- Pairwise kernel (Liao & Noble, 2003): the dictionary is the training set, the similarity is a classical measure of similarity between sequences.

Outline

Probabilistic models for sequences

Probabilistic modeling of biological sequences is older than kernel designs. Important models include HMM for protein sequences, SCFG for RNA sequences.



Parametric model

A model is a family of distribution

$$\{P_\theta, \theta \in \Theta \subset \mathbb{R}^m\} \subset \mathcal{M}_1^+(\mathcal{X})$$

Fisher kernel

Definition

- Fix a parameter $\theta_0 \in \Theta$ (e.g., by maximum likelihood over a training set of sequences)
- For each sequence $\mathbf{x}$, compute the Fisher score vector:

$$\Phi_{\theta_0}(\mathbf{x}) = \nabla_\theta \log P_\theta(\mathbf{x})|_{\theta=\theta_0} \ .$$

- Form the kernel (Jaakkola et al., 1998):

$$K\left(\mathbf{x}, \mathbf{x}'\right) = \Phi_{\theta_0}(\mathbf{x})^\top I(\theta_0)^{-1} \Phi_{\theta_0}(\mathbf{x}') \ ,$$

where $I(\theta_0) = E_{\theta_0}\left[\Phi_{\theta_0}(\mathbf{x})\Phi_{\theta_0}(\mathbf{x})^\top\right]$ is the Fisher information matrix.

Fisher kernel properties

- The Fisher score describes how each parameter contributes to the process of generating a particular example
- The Fisher kernel is invariant under change of parametrization of the model
- A kernel classifier employing the Fisher kernel derived from a model that contains the label as a latent variable is, asymptotically, at least as good a classifier as the MAP labelling based on the model (Jaakkola and Haussler, 1998).
- A variant of the Fisher kernel (called the Tangent of Posterior kernel) can also improve over the direct posterior classification by helping to correct the effect of estimation errors in the parameter (Tsuda et al., 2002).

Fisher kernel in practice

- $\Phi_{\theta_0}(\mathbf{x})$ can be computed explicitly for many models (e.g., HMMs)
- $I(\theta_0)$ is often replaced by the identity matrix
- Several different models (i.e., different θ_0) can be trained and combined
- Feature vectors are explicitly computed

Mutual information kernels

Definition

- Chose a prior $w(d\theta)$ on the measurable set Θ
- Form the kernel (Seeger, 2002):

$$K\left(\mathbf{x},\mathbf{x}'\right)=\int_{\theta\in\Theta}P_\theta(\mathbf{x})P_\theta(\mathbf{x}')w(d\theta)\ .$$

- No explicit computation of a finite-dimensional feature vector
- $K\left(\mathbf{x},\mathbf{x}'\right)=<\phi\left(\mathbf{x}\right),\phi\left(\mathbf{x}'\right)>_{L_2(w)}$ with

$$\phi\left(\mathbf{x}\right)=\left(P_\theta\left(\mathbf{x}\right)\right)_{\theta\in\Theta}\ .$$

Example: coin toss

- Let $P_\theta(X = 1) = \theta$ and $P_\theta(X = 0) = 1 - \theta$ a model for random coin toss, with $\theta \in [0, 1]$.
- Let $d\theta$ be the Lebesgue measure on $[0, 1]$
- The mutual information kernel between $\mathbf{x} = 001$ and $\mathbf{x}' = 1010$ is:

$$\begin{cases} P_\theta(\mathbf{x}) &= \theta(1-\theta)^2 \ , \\ P_\theta(\mathbf{x}') &= \theta^2(1-\theta)^2 \ , \end{cases}$$

$$K(\mathbf{x}, \mathbf{x}') = \int_0^1 \theta^3 (1-\theta)^4 \, d\theta = \frac{3!4!}{8!} = \frac{1}{280} \ .$$

Context-tree model

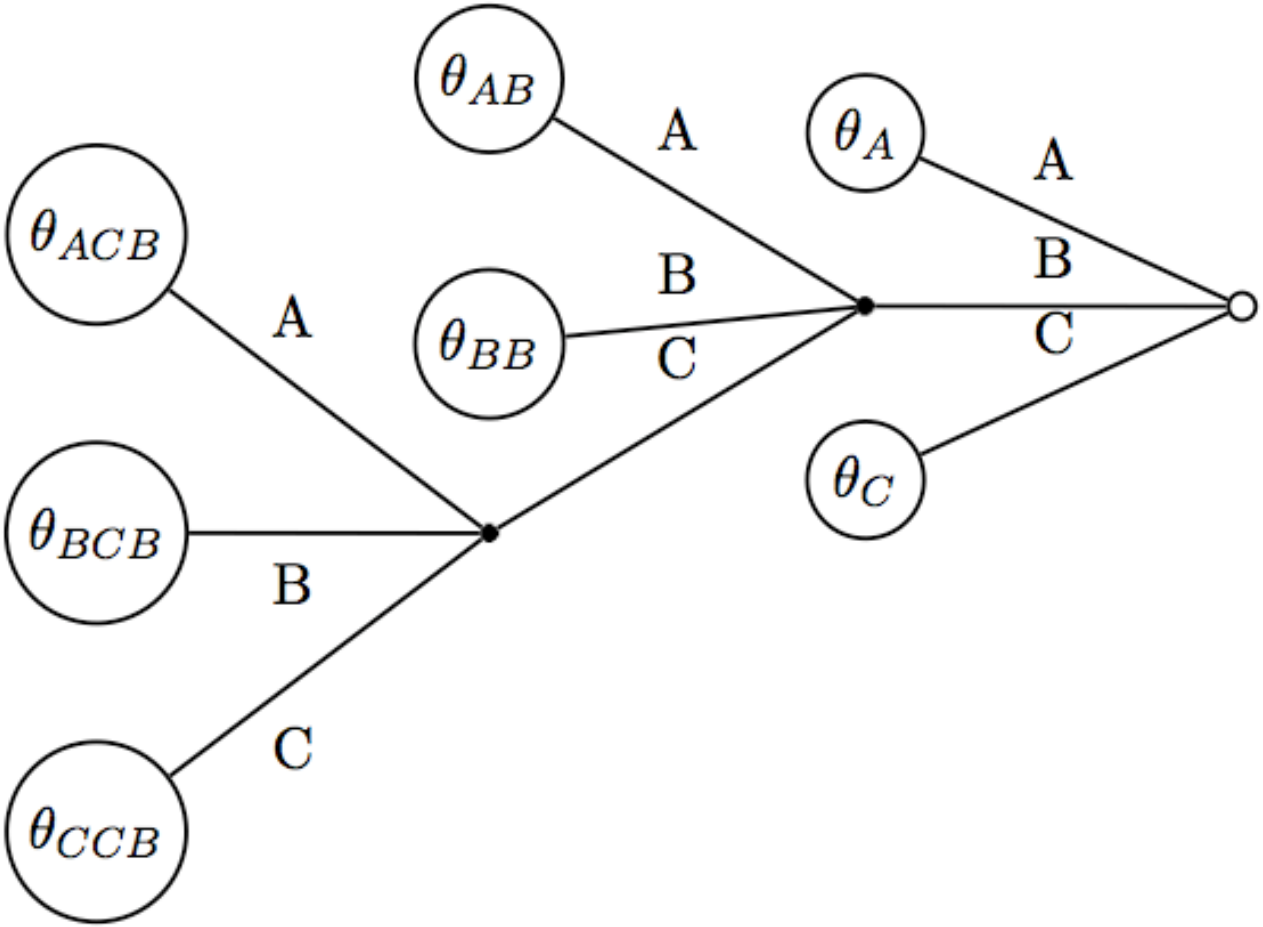
Definition

A context-tree model is a variable-memory Markov chain:

$$P_{\mathcal{D},\theta}(\mathbf{x}) = P_{\mathcal{D},\theta}(x_1 \dots x_D) \prod_{i=D+1}^{n} P_{\mathcal{D},\theta}(x_i \mid x_{i-D} \dots x_{i-1})$$

- $\mathcal{D}$ is a suffix tree
- $\theta \in \Sigma^{\mathcal{D}}$ is a set of conditional probabilities (multinomials)

Context-tree model: example



$$P(AABACBACC) = P(AAB)\theta_{AB}(A)\theta_A(C)\theta_C(B)\theta_{ACB}(A)\theta_A(C)\theta_C(A) \ .$$

The context-tree kernel

Theorem (Cuturi et al., 2004)

- *For particular choices of priors, the context-tree kernel:*

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathcal{D}} \int_{\theta \in \Sigma^{\mathcal{D}}} P_{\mathcal{D},\theta}(\mathbf{x}) P_{\mathcal{D},\theta}(\mathbf{x}') w(d\theta|\mathcal{D}) \pi(\mathcal{D})$$

  can be computed in $O(|\mathbf{x}| + |\mathbf{x}'|)$ with a variant of the Context-Tree Weighting algorithm.

- *This is a valid mutual information kernel.*
- *The similarity is related to information-theoretical measure of mutual information between strings.*

Marginalized kernels

Definition

- For any observed data $\mathbf{x} \in \mathcal{X}$, let a latent variable $\mathbf{y} \in \mathcal{Y}$ be associated probabilistically through a conditional probability $P_{\mathbf{x}}(d\mathbf{y})$.
- Let $K_{\mathcal{Z}}$ be a kernel for the complete data $\mathbf{z} = (\mathbf{x}, \mathbf{y})$
- Then the following kernel is a valid kernel on $\mathcal{X}$, called a marginalized kernel (Kin et al., 2002):

$$\begin{aligned} K_{\mathcal{X}}\left(\mathbf{x}, \mathbf{x}'\right) &:= E_{P_{\mathbf{x}}(d\mathbf{y}) \times P_{\mathbf{x}'}(d\mathbf{y}')} K_{\mathcal{Z}}\left(\mathbf{z}, \mathbf{z}'\right) \\ &= \int \int K_{\mathcal{Z}}\left((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')\right) P_{\mathbf{x}}(d\mathbf{y}) P_{\mathbf{x}'}\left(d\mathbf{y}'\right) \ . \end{aligned}$$

Marginalized kernels: proof of positive definiteness

- $K_{\mathcal{Z}}$ is p.d. on $\mathcal{Z}$. Therefore there exists a Hilbert space $\mathcal{H}$ and $\Phi_{\mathcal{Z}} : \mathcal{Z} \to \mathcal{H}$ such that:

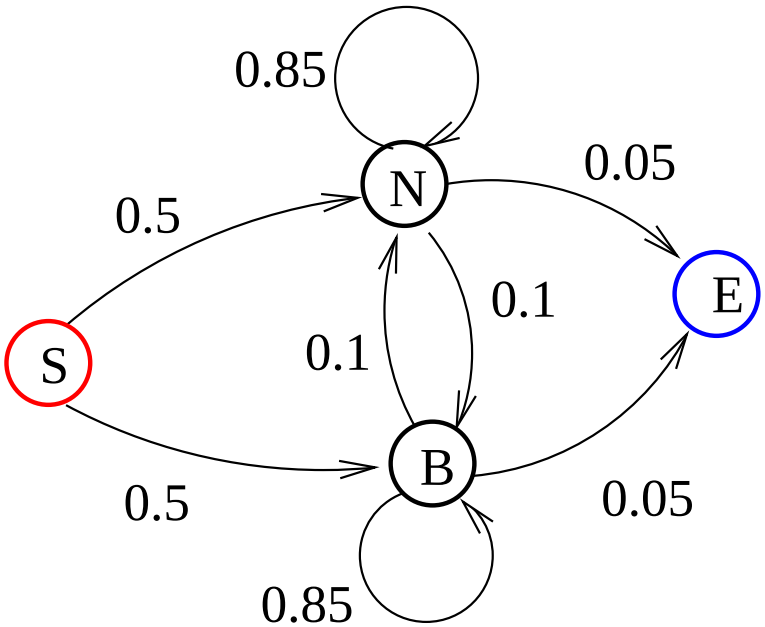
$$K_{\mathcal{Z}}\left(\mathbf{z}, \mathbf{z}'\right) = \left\langle \Phi_{\mathcal{Z}}\left(\mathbf{z}\right), \Phi_{\mathcal{Z}}\left(\mathbf{z}'\right) \right\rangle_{\mathcal{H}} .$$

- Marginalizing therefore gives:

$$\begin{aligned} K_{\mathcal{X}}\left(\mathbf{x}, \mathbf{x}'\right) &= E_{P_{\mathbf{x}}(d\mathbf{y}) \times P_{\mathbf{x}'}(d\mathbf{y}')} K_{\mathcal{Z}}\left(\mathbf{z}, \mathbf{z}'\right) \\ &= E_{P_{\mathbf{x}}(d\mathbf{y}) \times P_{\mathbf{x}'}(d\mathbf{y}')} \left\langle \Phi_{\mathcal{Z}}\left(\mathbf{z}\right), \Phi_{\mathcal{Z}}\left(\mathbf{z}'\right) \right\rangle_{\mathcal{H}} \\ &= \left\langle E_{P_{\mathbf{x}}(d\mathbf{y})} \Phi_{\mathcal{Z}}\left(\mathbf{z}\right), E_{P_{\mathbf{x}}(d\mathbf{y}')} \Phi_{\mathcal{Z}}\left(\mathbf{z}'\right) \right\rangle_{\mathcal{H}} , \end{aligned}$$

therefore $K_{\mathcal{X}}$ is p.d. on $\mathcal{X}$. $\square$

Example: HMM for normal/biased coin toss



- Normal (N) and biased (B) coins (not observed)

- Observed output are 0/1 with probabilities:

$$\begin{cases} \pi(0|N) = 1 - \pi(1|N) = 0.5, \\ \pi(0|B) = 1 - \pi(1|B) = 0.8. \end{cases}$$

- Example of realization (complete data):

```
NNNNNBBBBBBBBBNNNNNNNNNNNNBBBBBB
1001011101111010010111001111011
```

1-spectrum kernel on complete data

- If both $\mathbf{x} \in \mathcal{A}^*$ and $\mathbf{y} \in \mathcal{S}^*$ were observed, we might rather use the 1-spectrum kernel on the complete data $\mathbf{z} = (\mathbf{x}, \mathbf{y})$:

$$K_{\mathcal{Z}}\left(\mathbf{z}, \mathbf{z}'\right) = \sum_{(a,s) \in \mathcal{A} \times \mathcal{S}} n_{a,s}\left(\mathbf{z}\right) n_{a,s}\left(\mathbf{z}\right),$$

  where $n_{a,s}\left(\mathbf{x}, \mathbf{y}\right)$ for $a = 0, 1$ and $s = N, B$ is the number of occurrences of s in $\mathbf{y}$ which emit a in $\mathbf{x}$.

- Example:

$$\begin{aligned} \mathbf{z} &= 1001011101111010010111001111011, \\ \mathbf{z}' &= 0011010110011111011010111101100101, \end{aligned}$$

$$\begin{aligned} K_{\mathcal{Z}}\left(\mathbf{z}, \mathbf{z}'\right) &= n_0\left(\mathbf{z}\right) n_0\left(\mathbf{z}'\right) + n_0\left(\mathbf{z}\right) n_0\left(\mathbf{z}'\right) + n_1\left(\mathbf{z}\right) n_1\left(\mathbf{z}'\right) + n_1\left(\mathbf{z}\right) n_1\left(\mathbf{z}\right. \\ &= 7 \times 15 + 9 \times 12 + 13 \times 6 + 2 \times 1 = 293. \end{aligned}$$

1-spectrum marginalized kernel on observed data

- The marginalized kernel for observed data is:

$$K_{\mathcal{X}}\left(\mathbf{x},\mathbf{x}'\right) = \sum_{\mathbf{y},\mathbf{y}'\in\mathcal{S}^*} K_{\mathcal{Z}}\left(\left(\mathbf{x},\mathbf{y}\right),\left(\mathbf{x},\mathbf{y}\right)\right) P\left(\mathbf{y}|\mathbf{x}\right) P\left(\mathbf{y}'|\mathbf{x}'\right)$$
$$= \sum_{(a,s)\in\mathcal{A}\times\mathcal{S}} \Phi_{a,s}\left(\mathbf{x}\right)\Phi_{a,s}\left(\mathbf{x}'\right),$$

with

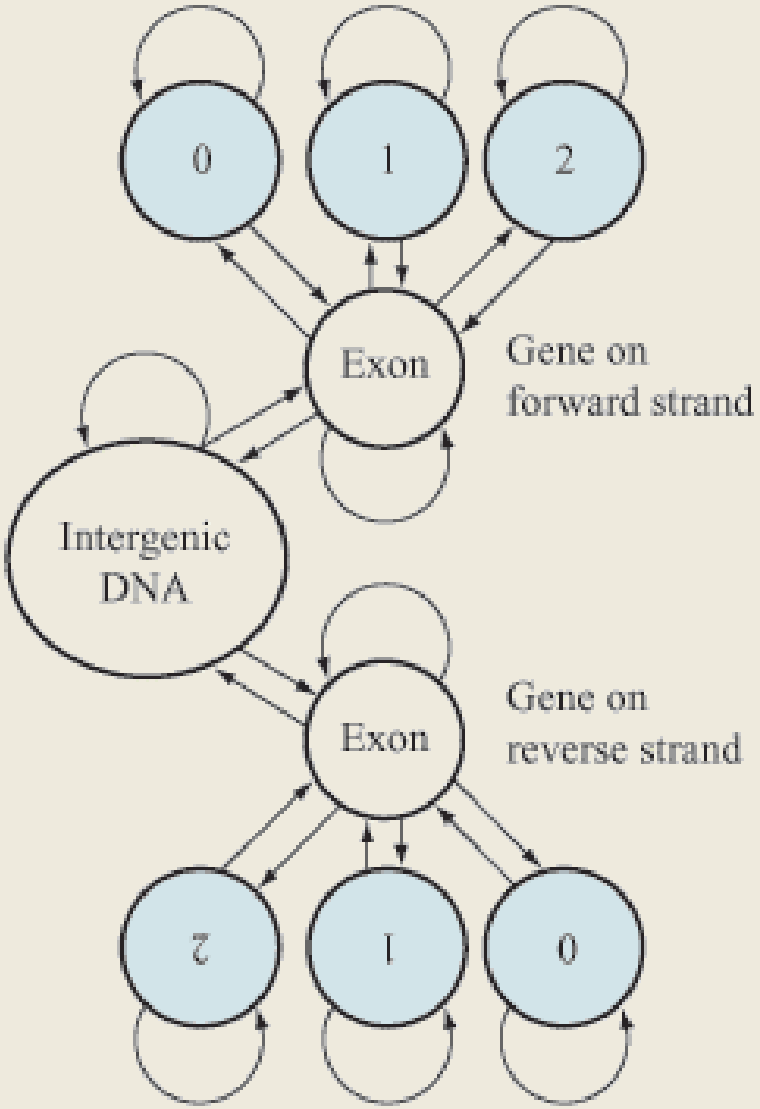
$$\Phi_{a,s}\left(\mathbf{x}\right) = \sum_{\mathbf{y}\in\mathcal{S}^*} P\left(\mathbf{y}|\mathbf{x}\right) n_{a,s}\left(\mathbf{x},\mathbf{y}\right)$$

Computation of the 1-spectrum marginalized kernel

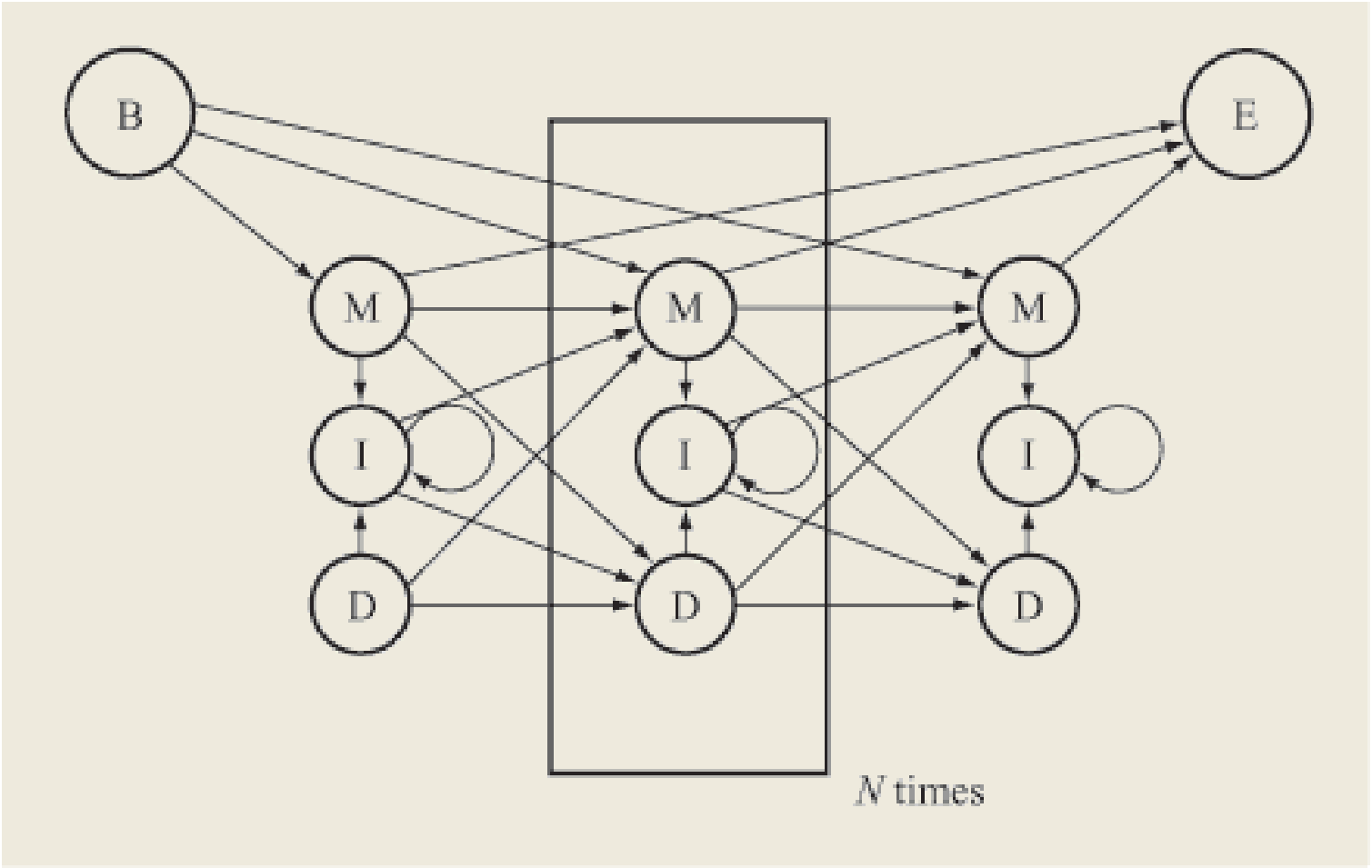
$$
\begin{aligned}
\Phi_{a,s}(\mathbf{x}) &= \sum_{\mathbf{y}\in\mathcal{S}^*} P(\mathbf{y}|\mathbf{x})\, n_{a,s}(\mathbf{x},\mathbf{y}) \\
&= \sum_{\mathbf{y}\in\mathcal{S}^*} P(\mathbf{y}|\mathbf{x}) \left\{ \sum_{i=1}^{n} \delta(x_i, a)\, \delta(y_i, s) \right\} \\
&= \sum_{i=1}^{n} \delta(x_i, a) \left\{ \sum_{\mathbf{y}\in\mathcal{S}^*} P(\mathbf{y}|\mathbf{x})\, \delta(y_i, s) \right\} \\
&= \sum_{i=1}^{n} \delta(x_i, a)\, P(y_i = s|\mathbf{x})\,.
\end{aligned}
$$

and $P(y_i = s|\mathbf{x})$ can be computed efficiently by forward-backward algorithm!

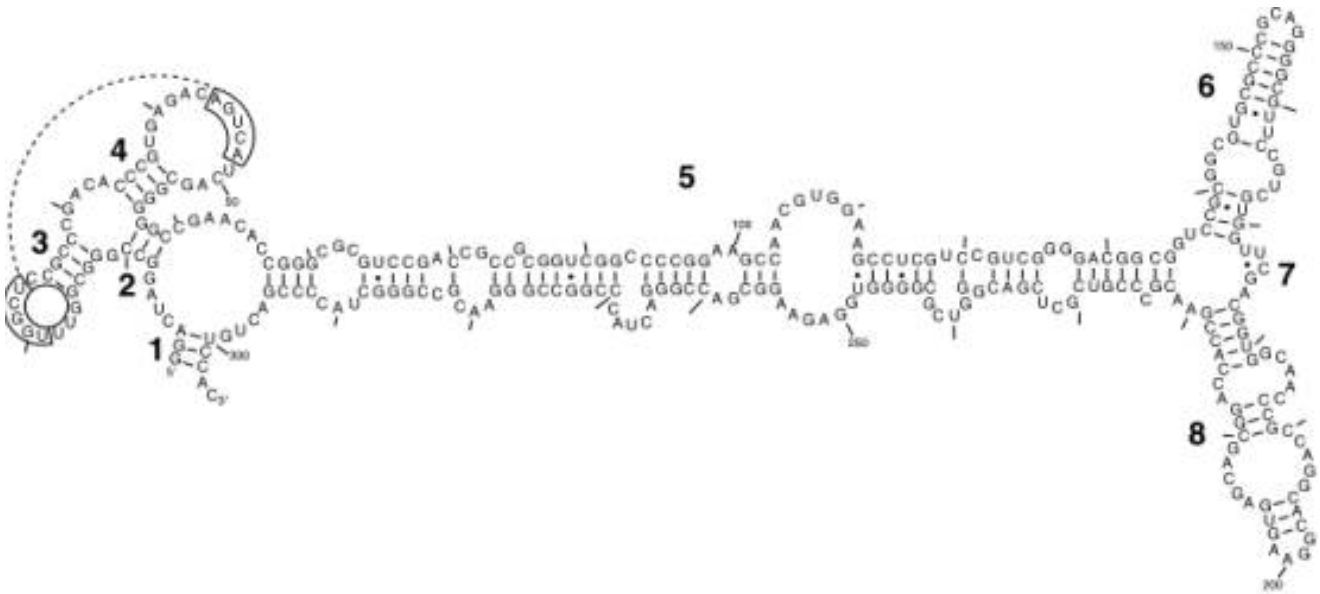
HMM example (DNA)



HMM example (protein)



SCFG for RNA sequences



SFCG rules

- $S \rightarrow SS$
- $S \rightarrow aSa$
- $S \rightarrow aS$
- $S \rightarrow a$

Marginalized kernel (Kin et al., 2002)

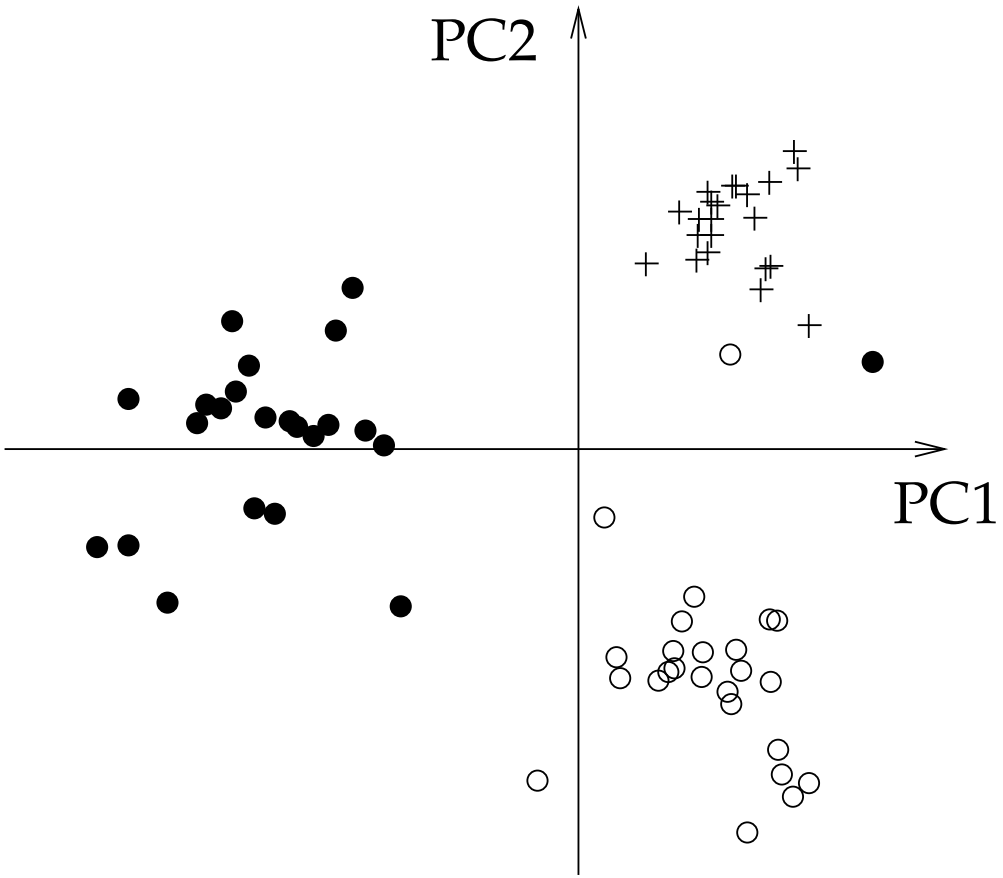
- Feature: number of occurrences of each (base,state) combination
- Marginalization using classical inside/outside algorithm

Marginalized kernels in practice

Examples

- Spectrum kernel on the hidden states of a HMM for protein sequences (Tsuda et al., 2002)
- Kernels for RNA sequences based on SCFG (Kin et al., 2002)
- Kernels for graphs based on random walks on graphs (Kashima et al., 2004)
- Kernels for multiple alignments based on phylogenetic models (Vert et al., 2005)

Marginalized kernels: example



A set of 74 human tRNA sequences is analyzed using a kernel for sequences (the second-order marginalized kernel based on SCFG). This set of tRNAs contains three classes, called Ala-AGC (*white circles*), Asn-GTT (*black circles*) and Cys-GCA (*plus symbols*) (from Tsuda et al., 2003).

Outline

Sequence alignment

Motivation

How to compare 2 sequences?

$$\mathbf{x}_1 = \texttt{CGGSLIAMMWFGV}$$
$$\mathbf{x}_2 = \texttt{CLIVMMNRLMWFGV}$$

Find a good alignment:

```
CGGSLIAMM----WFGV
|...|||||....||||
C---LIVMMNRLMWFGV
```

Alignment score

In order to quantify the relevance of an alignment π, define:

- a substitution matrix $S \in \mathbb{R}^{\mathcal{A} \times \mathcal{A}}$
- a gap penalty function $g : \mathbb{N} \longrightarrow \mathbb{R}$

Any alignment is then scored as follows

```
CGGSLIAMM----WFGV
|...|||||....||||
C---LIVMMNRLMWFGV
```

$$s_{S,g}(\pi) = S(C,C) + S(L,L) + S(I,I) + S(A,V) + 2S(M,M) + S(W,W) + S(F,F) + S(G,G) + S(V,V) - g(3) - g(4)$$

Local alignment kernel

Smith-Waterman score

- The widely-used Smith-Waterman local alignment score is defined by:

$$SW_{S,g}(\mathbf{x},\mathbf{y}) := \max_{\pi \in \Pi(\mathbf{x},\mathbf{y})} s_{S,g}(\pi).$$

- It is symmetric, but not positive definite...

LA kernel

The local alignment kernel:

$$K_{LA}^{(\beta)}(\mathbf{x},\mathbf{y}) = \sum_{\pi \in \Pi(\mathbf{x},\mathbf{y})} \exp\left(\beta s_{S,g}(\mathbf{x},\mathbf{y},\pi)\right),$$

is symmetric positive definite.

Local alignment kernel

Smith-Waterman score

- The widely-used Smith-Waterman local alignment score is defined by:

$$SW_{S,g}(\mathbf{x},\mathbf{y}) := \max_{\pi \in \Pi(\mathbf{x},\mathbf{y})} s_{S,g}(\pi).$$

- It is symmetric, but not positive definite...

LA kernel

The local alignment kernel:

$$K_{LA}^{(\beta)}(\mathbf{x},\mathbf{y}) = \sum_{\pi \in \Pi(\mathbf{x},\mathbf{y})} \exp\left(\beta s_{S,g}(\mathbf{x},\mathbf{y},\pi)\right),$$

is symmetric positive definite.

LA kernel is p.d.: proof (1/11)

Lemma

- If K_1 and K_2 are p.d. kernels, then:

$$K_1 + K_2,$$
$$K_1 K_2, \text{ and}$$
$$cK_1, \text{ for } c \geq 0,$$

  are also p.d. kernels
- If $(K_i)_{i \geq 1}$ is a sequence of p.d. kernels that converges pointwisely to a function K:

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

  then K is also a p.d. kernel.

LA kernel is p.d.: proof (2/11)

Proof of lemma

Let A and B be $n \times n$ positive semidefinite matrices. By diagonalization of A:

$$A_{i,j} = \sum_{p=1}^{n} f_p(i) f_p(j)$$

for some vectors $f_1, \ldots, f_n$. Then, for any $\alpha \in \mathbb{R}^n$:

$$\sum_{i,j=1}^{n} \alpha_i \alpha_j A_{i,j} B_{i,j} = \sum_{p=1}^{n} \sum_{i,j=1}^{n} \alpha_i f_p(i) \alpha_j f_p(j) B_{i,j} \geq 0.$$

The matrix $C_{i,j} = A_{i,j} B_{i,j}$ is therefore p.d. Other properties are obvious from definition. $\square$

LA kernel is p.d.: proof (3/11)

Lemma (direct sum and product of kernels)

Let $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$. Let K_1 be a p.d. kernel on $\mathcal{X}_1$, and K_2 be a p.d. kernel on $\mathcal{X}_2$. Then the following functions are p.d. kernels on $\mathcal{X}$:

- the direct sum,

$$K\left((\mathbf{x}_1, \mathbf{x}_2), (\mathbf{y}_1, \mathbf{y}_2)\right) = K_1\left(\mathbf{x}_1, \mathbf{y}_1\right) + K_2\left(\mathbf{x}_2, \mathbf{y}_2\right),$$

- The direct product:

$$K\left((\mathbf{x}_1, \mathbf{x}_2), (\mathbf{y}_1, \mathbf{y}_2)\right) = K_1\left(\mathbf{x}_1, \mathbf{y}_1\right) K_2\left(\mathbf{x}_2, \mathbf{y}_2\right).$$

LA kernel is p.d.: proof (4/11)

Proof of lemma

If K_1 is a p.d. kernel, let $\Phi_1 : \mathcal{X}_1 \mapsto \mathcal{H}$ be such that:

$$K_1(\mathbf{x}_1, \mathbf{y}_1) = \langle \Phi_1(\mathbf{x}_1), \Phi_1(\mathbf{y}_1) \rangle_{\mathcal{H}}.$$

Let $\Phi : \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{H}$ be defined by:

$$\Phi((\mathbf{x}_1, \mathbf{x}_2)) = \Phi_1(\mathbf{x}_1).$$

Then for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ and $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2) \in \mathcal{X}$, we get

$$\langle \Phi((\mathbf{x}_1, \mathbf{x}_2)), \Phi((\mathbf{y}_1, \mathbf{y}_2)) \rangle_{\mathcal{H}} = K_1(\mathbf{x}_1, \mathbf{x}_2),$$

which shows that $K(\mathbf{x}, \mathbf{y}) := K_1(\mathbf{x}_1, \mathbf{y}_1)$ is p.d. on $\mathcal{X}_1 \times \mathcal{X}_2$. The lemma follows from the properties of sums and products of p.d. kernels. $\square$

LA kernel is p.d.: proof (5/11)

Lemma: kernel for sets

Let K be a p.d. kernel on $\mathcal{X}$, and let $\mathcal{P}(\mathcal{X})$ be the set of finite subsets of $\mathcal{X}$. Then the function K_P on $\mathcal{P}(\mathcal{X}) \times \mathcal{P}(\mathcal{X})$ defined by:

$$\forall A, B \in \mathcal{P}(\mathcal{X}), \quad K_P(A, B) := \sum_{\mathbf{x} \in A} \sum_{\mathbf{y} \in B} K(\mathbf{x}, \mathbf{y})$$

is a p.d. kernel on $\mathcal{P}(\mathcal{X})$.

LA kernel is p.d.: proof (6/11)

Proof of lemma

Let $\Phi : \mathcal{X} \mapsto \mathcal{H}$ be such that

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

Then, for $A, B \in \mathcal{P}(\mathcal{X})$, we get:

$$\begin{aligned} K_P(A, B) &= \sum_{\mathbf{x} \in A} \sum_{\mathbf{y} \in B} \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle_{\mathcal{H}} \\ &= \left\langle \sum_{\mathbf{x} \in A} \Phi(\mathbf{x}), \sum_{\mathbf{y} \in B} \Phi(\mathbf{y}) \right\rangle_{\mathcal{H}} \\ &= \langle \Phi_P(A), \Phi_P(B) \rangle_{\mathcal{H}}, \end{aligned}$$

with $\Phi_P(A) := \sum_{\mathbf{x} \in A} \Phi(\mathbf{x})$. $\square$

LA kernel is p.d.: proof (7/11)

Definition: Convolution kernel (Haussler, 1999)

Let K_1 and K_2 be two p.d. kernels for strings. The convolution of K_1 and K_2, denoted $K_1 \star K_2$, is defined for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ by:

$$K_1 \star K_2(\mathbf{x}, \mathbf{y}) := \sum_{\mathbf{x}_1\mathbf{x}_2=\mathbf{x}, \mathbf{y}_1\mathbf{y}_2=\mathbf{y}} K_1(\mathbf{x}_1, \mathbf{y}_1) K_2(\mathbf{x}_2, \mathbf{y}_2).$$

Lemma

If K_1 and K_2 are p.d. then $K_1 \star K_2$ is p.d..

LA kernel is p.d.: proof (8/11)

Proof of lemma

Let $\mathcal{X}$ be the set of finite-length strings. For $\mathbf{x} \in \mathcal{X}$, let

$$R(\mathbf{x}) = \{(\mathbf{x}_1, \mathbf{x}_2) \in \mathcal{X} \times \mathcal{X} : \mathbf{x} = \mathbf{x}_1 \mathbf{x}_2\} \subset \mathcal{X} \times \mathcal{X} \,.$$

We can then write

$$K_1 \star K_2(\mathbf{x}, \mathbf{y}) = \sum_{(\mathbf{x}_1, \mathbf{x}_2) \in R(\mathbf{x})} \sum_{(\mathbf{y}_1, \mathbf{y}_2) \in R(\mathbf{y})} K_1(\mathbf{x}_1, \mathbf{y}_1) K_2(\mathbf{x}_2, \mathbf{y}_2)$$

which is a p.d. kernel by the previous lemmas. $\square$

LA kernel is p.d.: proof (9/11)

3 basic string kernels

- The constant kernel:

$$K_0(\mathbf{x}, \mathbf{y}) := 1 .$$

- A kernel for letters:

$$K_a^{(\beta)}(\mathbf{x}, \mathbf{y}) := \begin{cases} 0 & \text{if } |\mathbf{x}| \neq 1 \text{ where } |\mathbf{y}| \neq 1 , \\ \exp(\beta S(\mathbf{x}, \mathbf{y})) & \text{otherwise} . \end{cases}$$

- A kernel for gaps:

$$K_g^{(\beta)}(\mathbf{x}, \mathbf{y}) = \exp\left[\beta\left(g(|\mathbf{x}|) + g(|\mathbf{x}|)\right)\right] .$$

LA kernel is p.d.: proof (10/11)

Remark

- $S : \mathcal{A}^2 \to \mathbb{R}$ is the similarity function between letters used in the alignment score. $K_a^{(\beta)}$ is only p.d. when the matrix:

$$\left(\exp\left(\beta s(a,b)\right)\right)_{(a,b)\in\mathcal{A}^2}$$

  is positive semidefinite (this is true for all β when s is conditionally p.d..
- g is the gap penalty function used in alignment score. The gap kernel is always p.d. (with no restriction on g) because it can be written as:

$$K_g^{(\beta)}(\mathbf{x},\mathbf{y}) = \exp\left(\beta g\left(|\,\mathbf{x}\,|\right)\right) \times \exp\left(\beta g\left(|\,\mathbf{y}\,|\right)\right) .$$

LA kernel is p.d.: proof (11/11)

Lemma

The local alignment kernel is a (limit) of convolution kernel:

$$K_{LA}^{(\beta)} = \sum_{n=0}^{\infty} K_0 \star \left(K_a^{(\beta)} \star K_g^{(\beta)} \right)^{(n-1)} \star K_a^{(\beta)} \star K_0.$$

As such it is p.d..

Proof (sketch)

- By induction on n (simple but long to write).
- See details in Vert et al. (2004).

LA kernel computation

- We assume an affine gap penalty:

$$\begin{cases} g(0) & = 0, \\ g(n) & = d + e(n-1) \text{ si } n \geq 1, \end{cases}$$

- The LA kernel can then be computed by dynamic programming by:

$$K_{LA}^{(\beta)}(\mathbf{x}, \mathbf{y}) = 1 + X_2(|\mathbf{x}|, |\mathbf{y}|) + Y_2(|\mathbf{x}|, |\mathbf{y}|) + M(|\mathbf{x}|, |\mathbf{y}|),$$

where $M(i,j), X(i,j), Y(i,j), X_2(i,j)$, and $Y_2(i,j)$ for $0 \leq i \leq |\mathbf{x}|$, and $0 \leq j \leq |\mathbf{y}|$ are defined recursively.

LA kernel is p.d.: proof (/)

Initialization

$$\begin{cases} M(i,0) = M(0,j) = 0, \\ X(i,0) = X(0,j) = 0, \\ Y(i,0) = Y(0,j) = 0, \\ X_2(i,0) = X_2(0,j) = 0, \\ Y_2(i,0) = Y_2(0,j) = 0, \end{cases}$$

LA kernel is p.d.: proof (/)

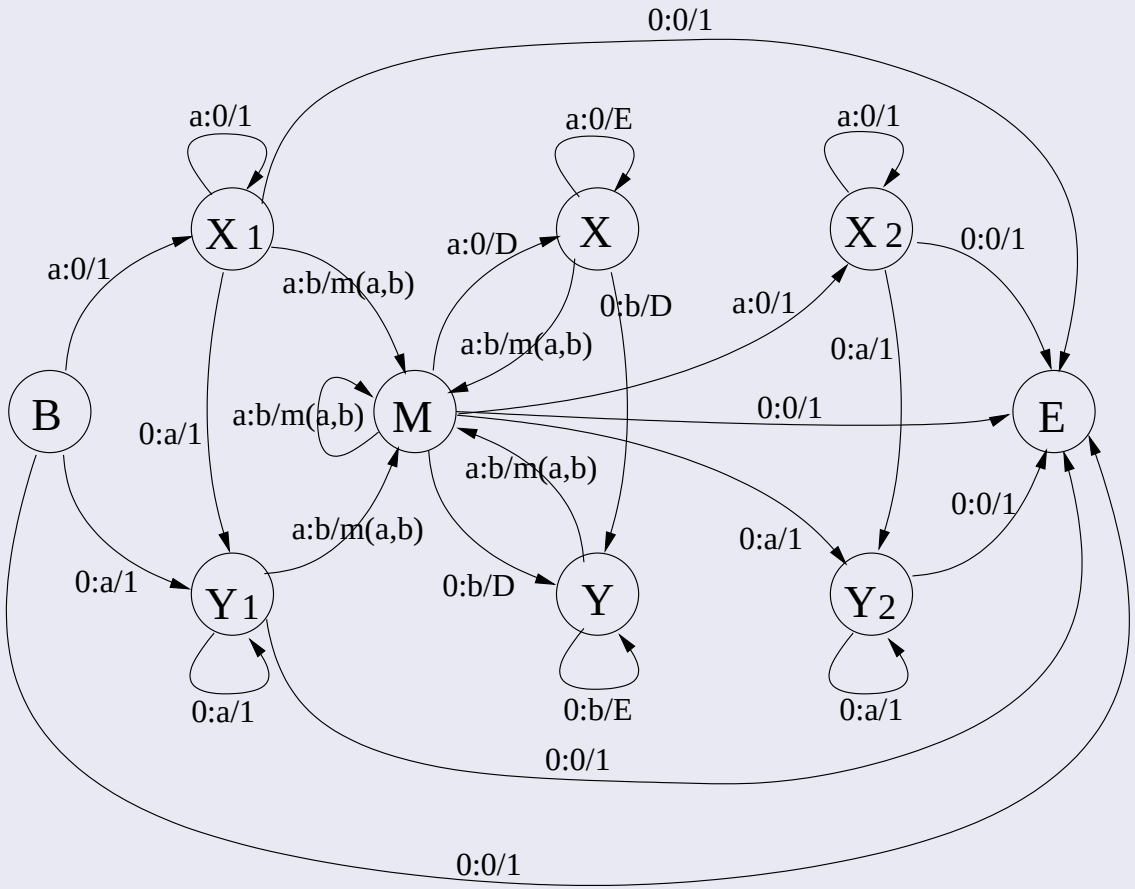
Recursion

For $i = 1, \ldots, |\mathbf{x}|$ and $j = 1, \ldots, |\mathbf{y}|$:

$$
\begin{cases}
M(i,j) & = \exp(\beta S(x_i, y_j)) \Big[1 + X(i-1, j-1) \\
& \qquad + Y(i-1, j-1) + M(i-1, j-1)\Big], \\
X(i,j) & = \exp(\beta d) M(i-1, j) + \exp(\beta e) X(i-1, j), \\
Y(i,j) & = \exp(\beta d) \left[M(i, j-1) + X(i, j-1)\right] \\
& \qquad + \exp(\beta e) Y(i, j-1), \\
X_2(i,j) & = M(i-1, j) + X_2(i-1, j), \\
Y_2(i,j) & = M(i, j-1) + X_2(i, j-1) + Y_2(i, j-1).
\end{cases}
$$

LA kernel in practice

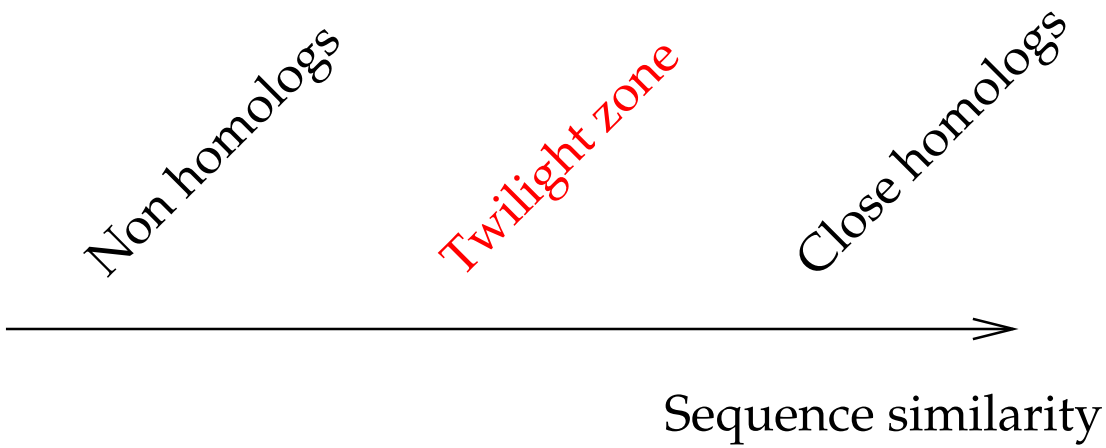
- Implementation by a finite-state transducer in $O(|\mathbf{x}| \times |\mathbf{x}'|)$



- In practice, values are too large (exponential scale) so taking its logarithm is a safer choice (but not p.d. anymore!)

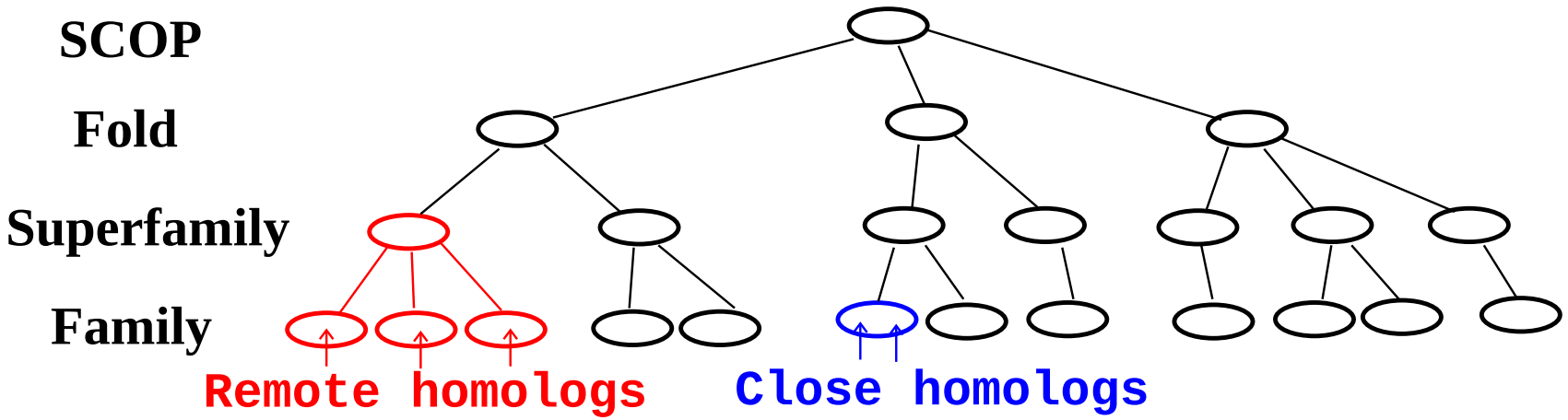
Outline

Remote homology



- Homologs have common ancestors
- Structures and functions are more conserved than sequences
- Remote homologs can not be detected by direct sequence comparison

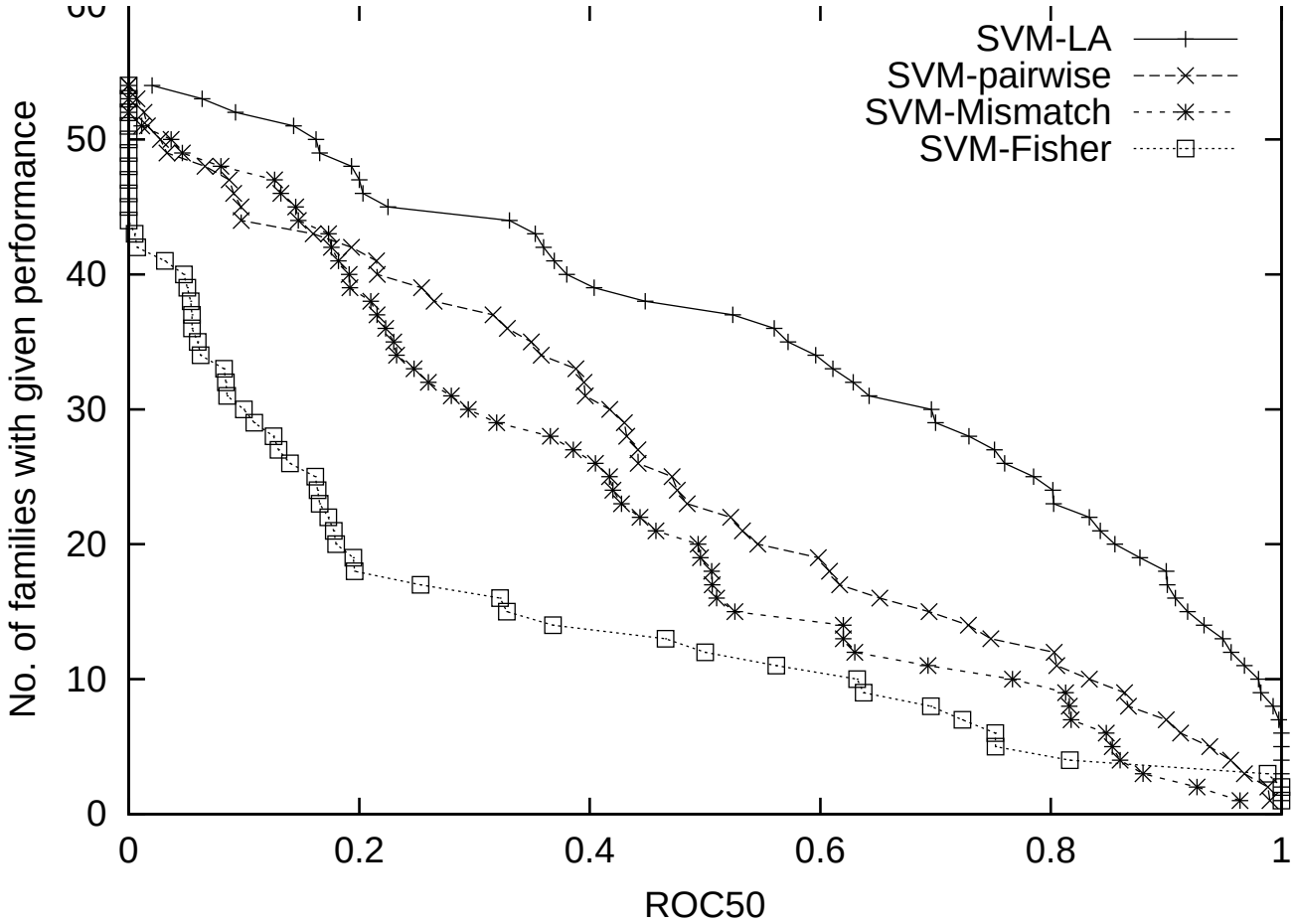
SCOP database



A benchmark experiment

- Goal: recognize directly the superfamily
- Training: for a sequence of interest, positive examples come from the same superfamily, but different families. Negative from other superfamilies.
- Test: predict the superfamily.

Difference in performance



Performance on the SCOP superfamily recognition benchmark (from Vert et al., 2004).

String kernels: Summary

- A variety of principles for string kernel design have been proposed.
- Good kernel design is important for each data and each task. Performance is not the only criterion.
- Still an art, although principled ways have started to emerge.
- Fast implementation with string algorithms is often possible.
- Their application goes well beyond computational biology.

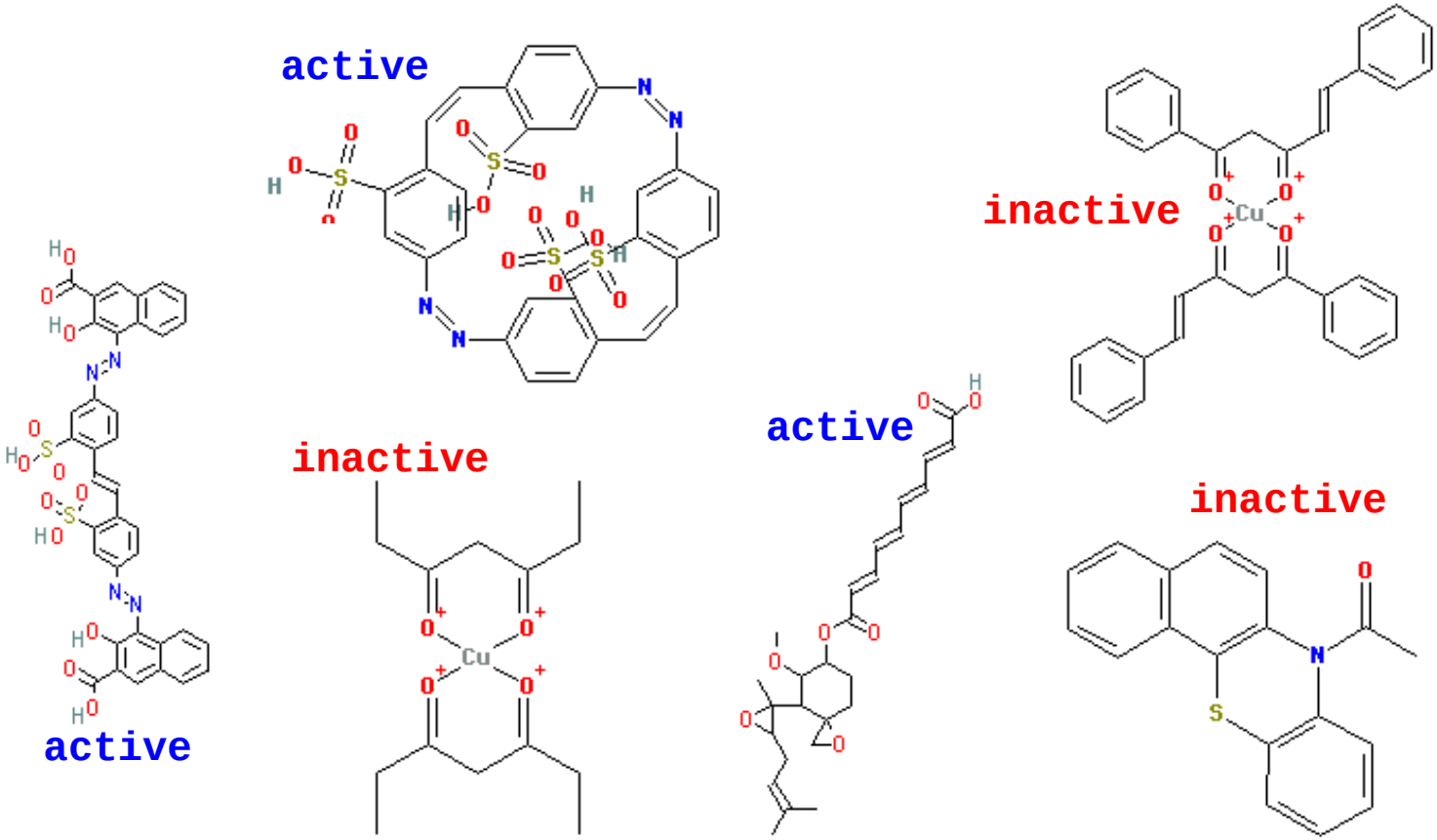
Outline

Part 6

Kernels for graphs

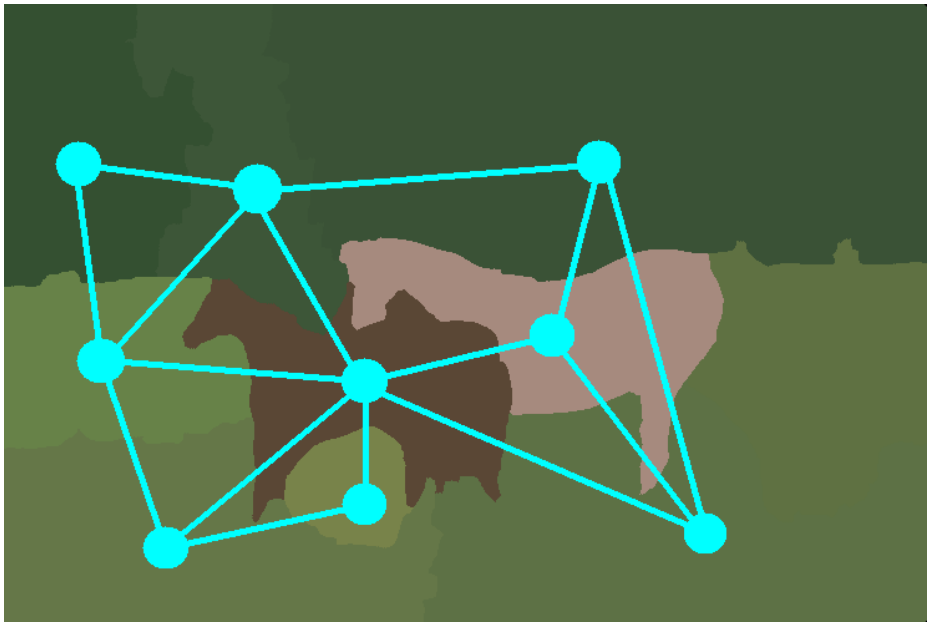
Outline

Virtual screening for drug discovery



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

Image retrieval and classification



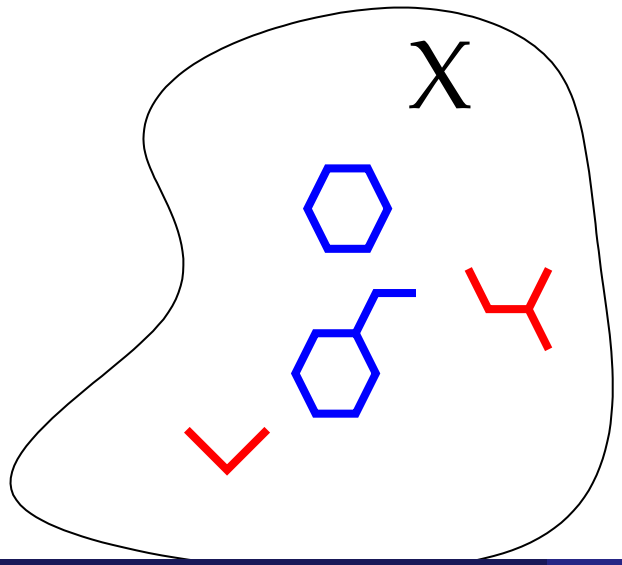
From Harchaoui and Bach (2007).

Our approach

1. Represent each graph x by a vector $\Phi(x) \in \mathcal{H}$, either explicitly or implicitly through the kernel

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

2. Use a linear method for classification in $\mathcal{H}$.

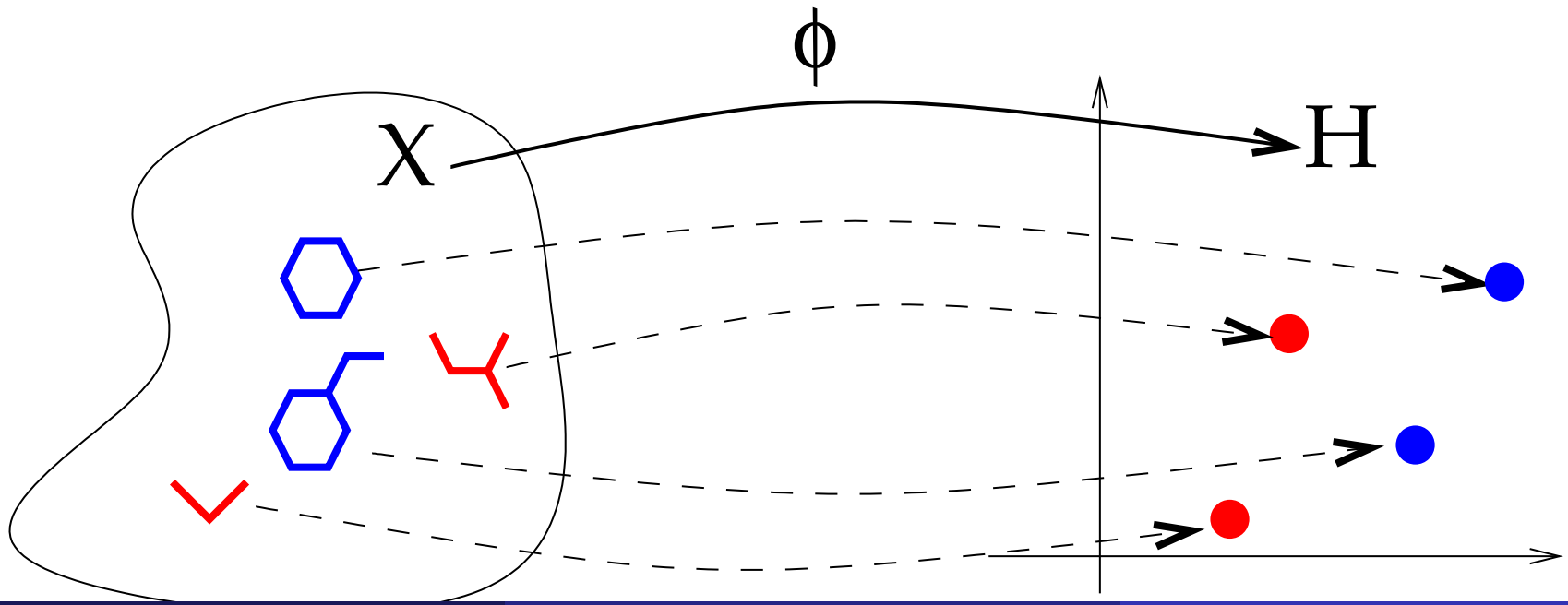


Our approach

1. Represent each graph x by a vector $\Phi(x) \in \mathcal{H}$, either explicitly or implicitly through the kernel

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

2. Use a linear method for classification in $\mathcal{H}$.

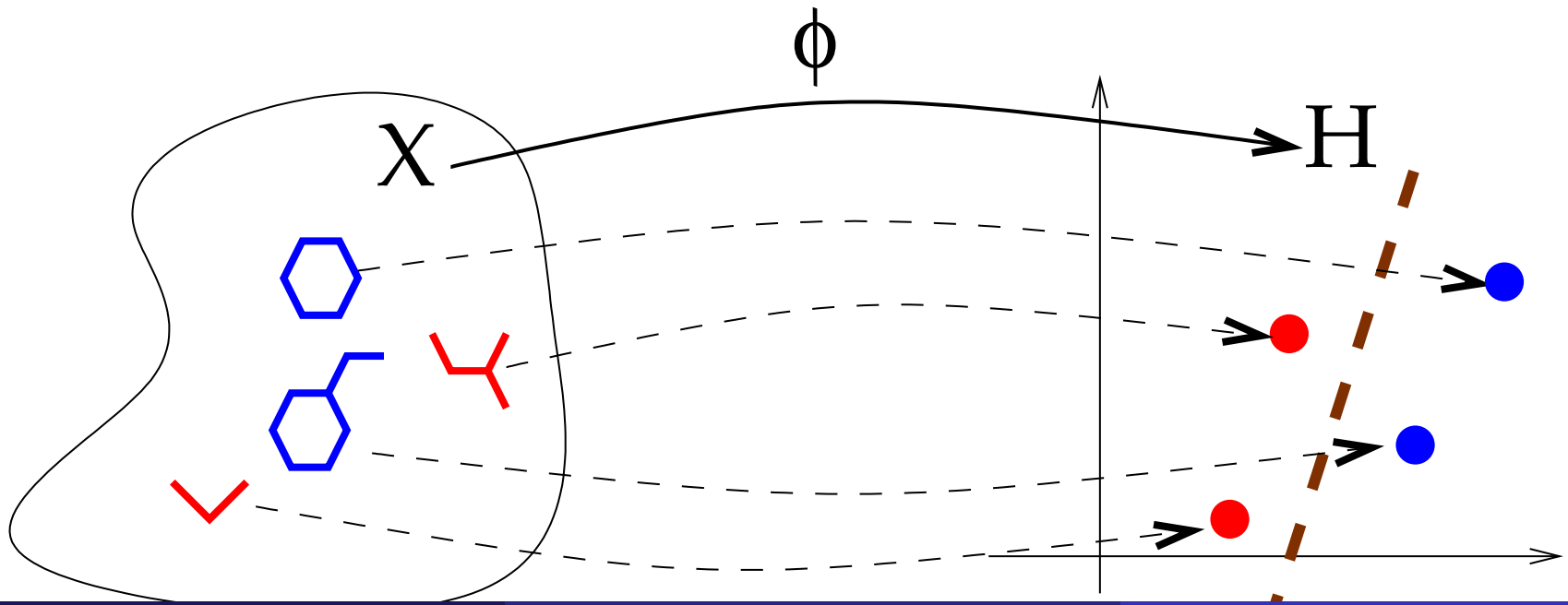


Our approach

1. Represent each graph x by a vector $\Phi(x) \in \mathcal{H}$, either explicitly or implicitly through the kernel

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

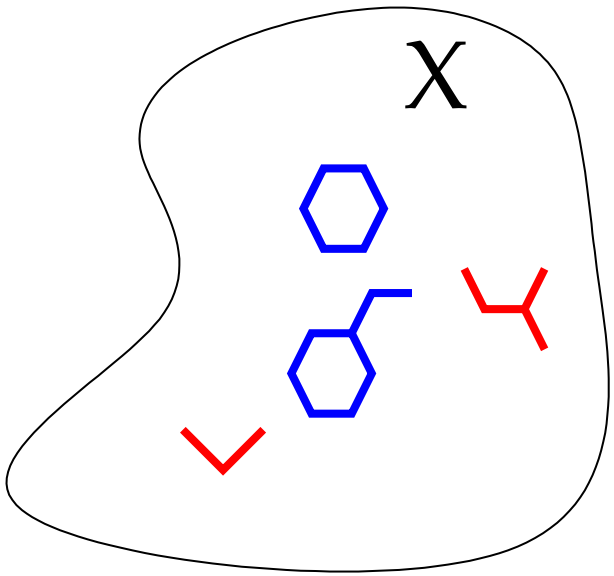
2. Use a linear method for classification in $\mathcal{H}$.



Outline

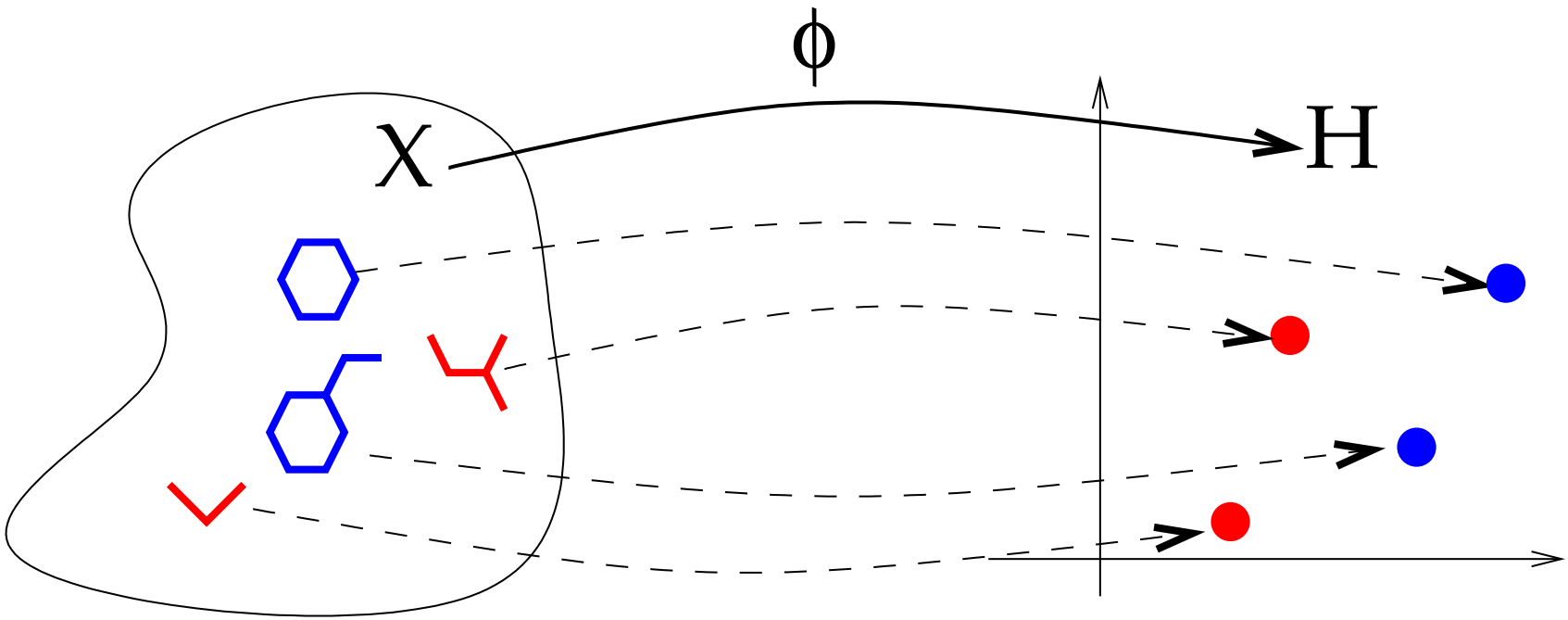
The approach

1. Represent explicitly each graph x by a vector of fixed dimension $\Phi(x) \in \mathbb{R}^p$.
2. Use an algorithm for regression or pattern recognition in $\mathbb{R}^p$.



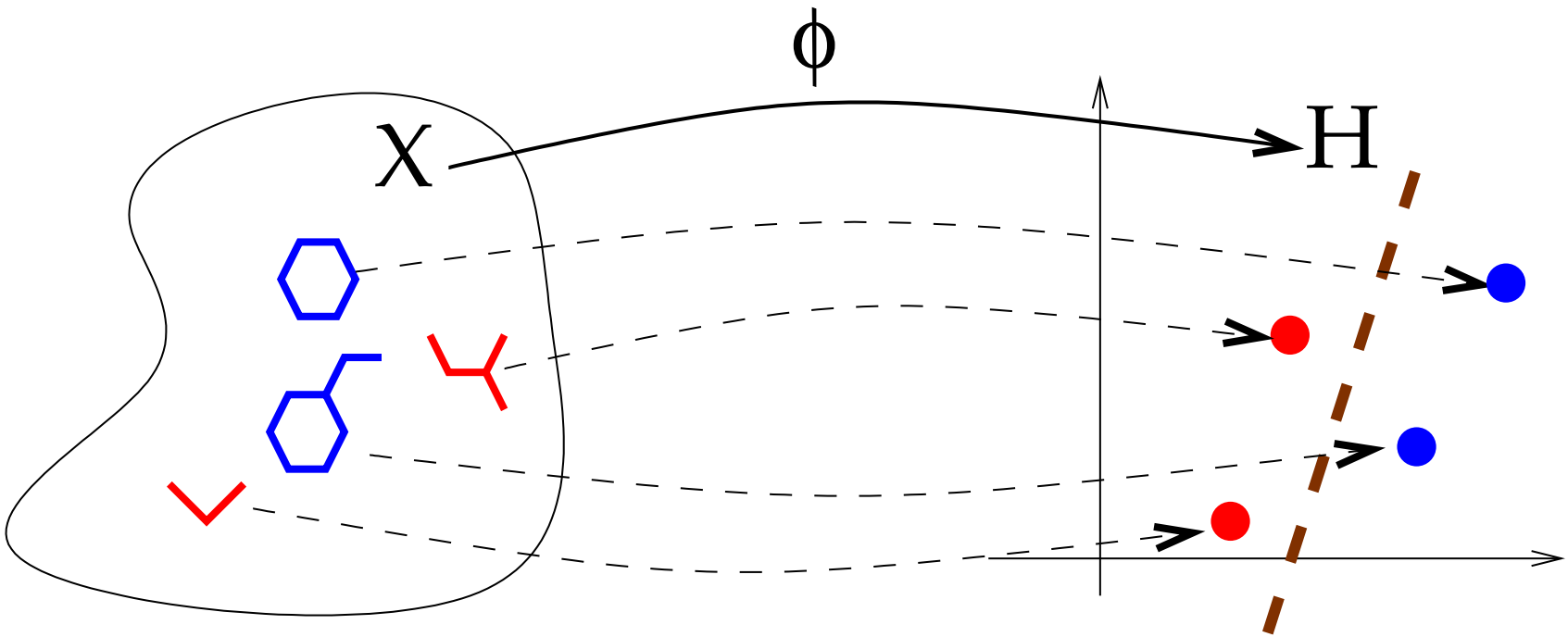
The approach

1. Represent explicitly each graph x by a vector of fixed dimension $\Phi(x) \in \mathbb{R}^p$.
2. Use an algorithm for regression or pattern recognition in $\mathbb{R}^p$.



The approach

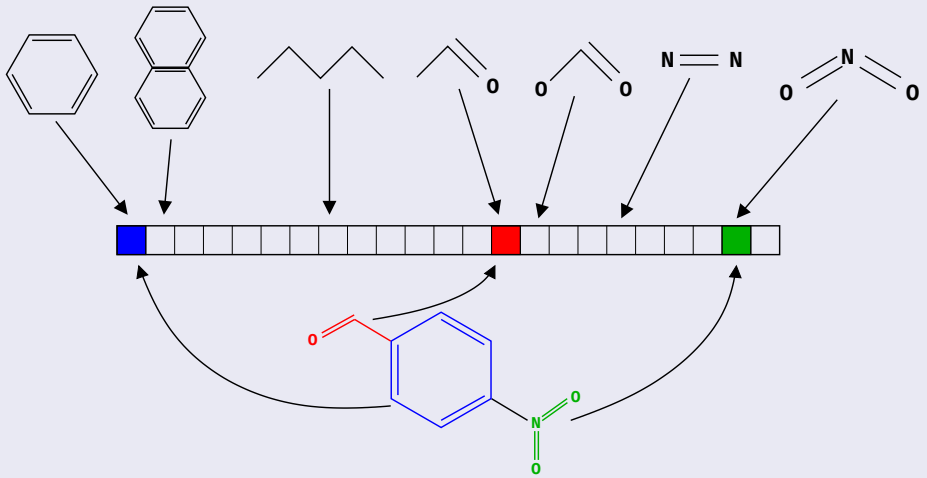
1. Represent explicitly each graph x by a vector of fixed dimension $\Phi(x) \in \mathbb{R}^p$.
2. Use an algorithm for regression or pattern recognition in $\mathbb{R}^p$.



Example

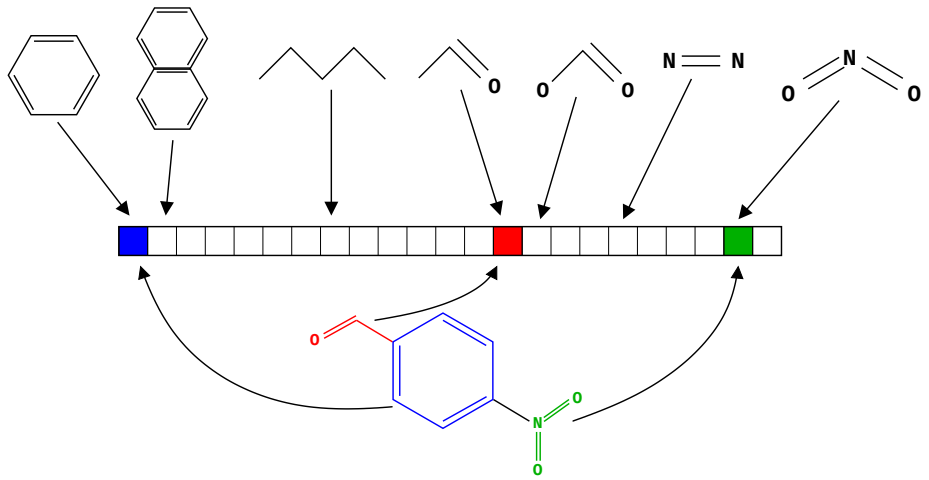
2D structural keys in chemoinformatics

- Index a molecule by a binary fingerprint defined by a limited set of pre-defined stuctures



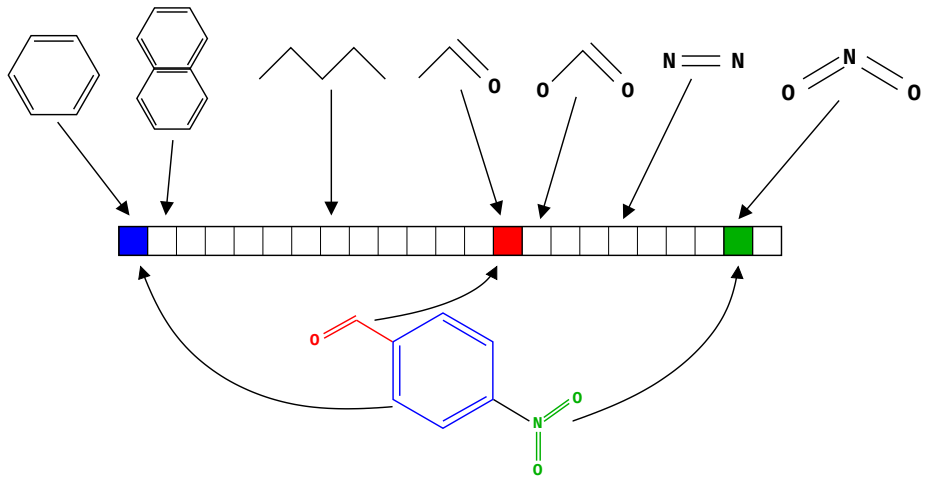
- Use a machine learning algorithms such as SVM, NN, PLS, decision tree, ...

Challenge: which descriptors (patterns)?



- Expressiveness: they should retain as much information as possible from the graph
- Computation : they should be fast to compute
- Large dimension of the vector representation: memory storage, speed, statistical issues

Indexing by substructures

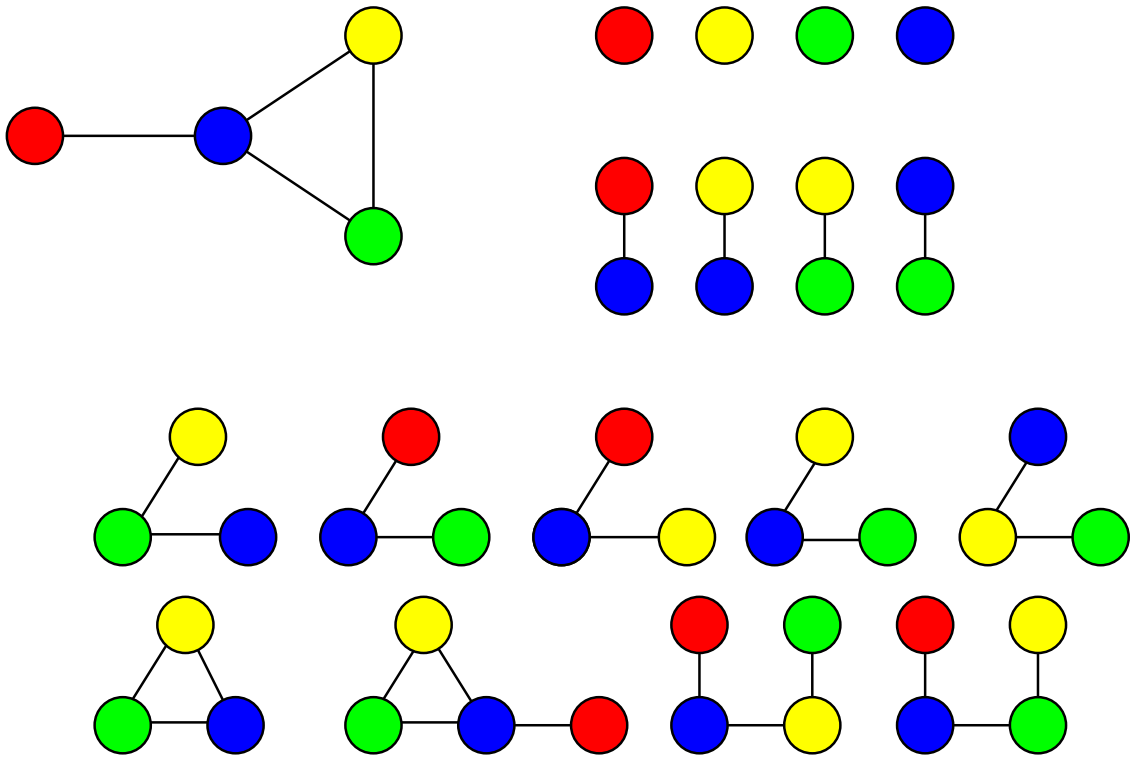


- Often we believe that the presence substructures are important predictive patterns
- Hence it makes sense to represent a graph by features that indicate the presence (or the number of occurrences) of particular substructures
- However, detecting the presence of particular substructures may be computationally challenging...

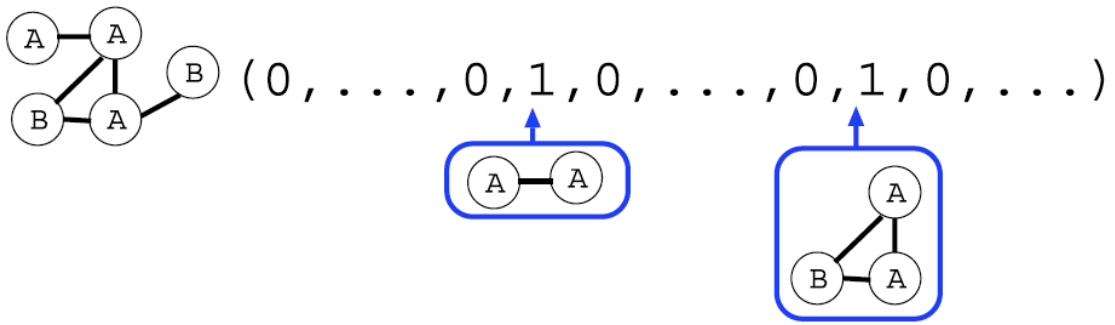
Subgraphs

Definition

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



Indexing by all subgraphs?



Theorem

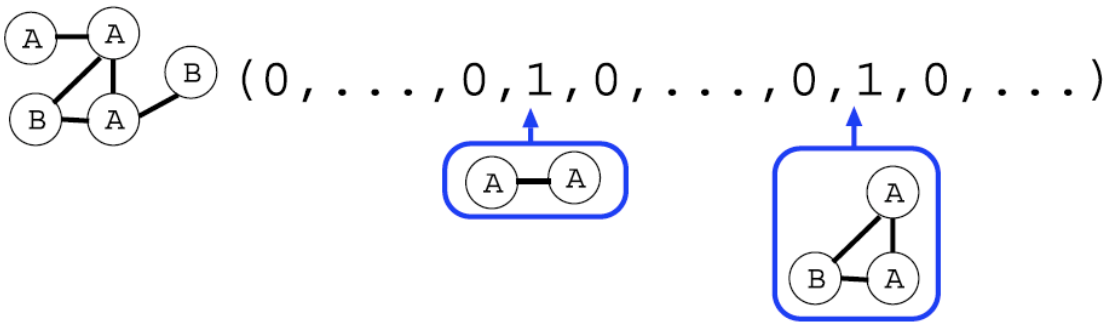
Computing all subgraph occurrences is NP-hard.

Proof.

- The linear graph of size n is a subgraph of a graph X with n vertices iff X has an Hamiltonian path
- The decision problem whether a graph has a Hamiltonian path is NP-complete.

□

Indexing by all subgraphs?



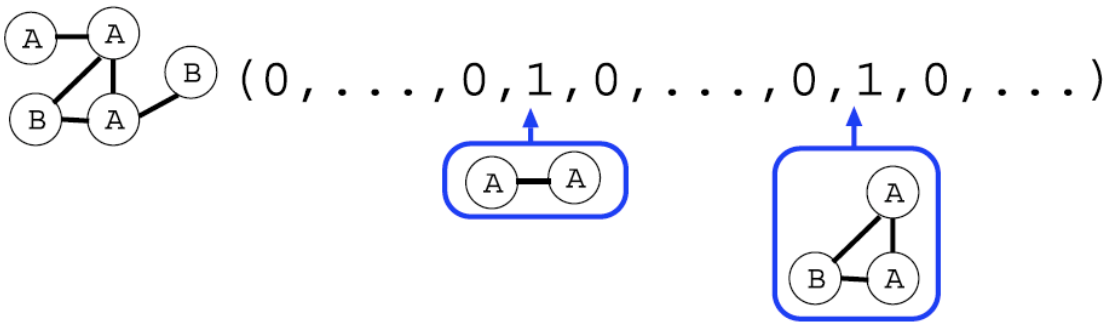
Theorem

Computing all subgraph occurrences is NP-hard.

Proof.

- The linear graph of size n is a subgraph of a graph X with n vertices iff X has an Hamiltonian path
- The decision problem whether a graph has a Hamiltonian path is NP-complete.

Indexing by all subgraphs?



Theorem

Computing all subgraph occurrences is NP-hard.

Proof.

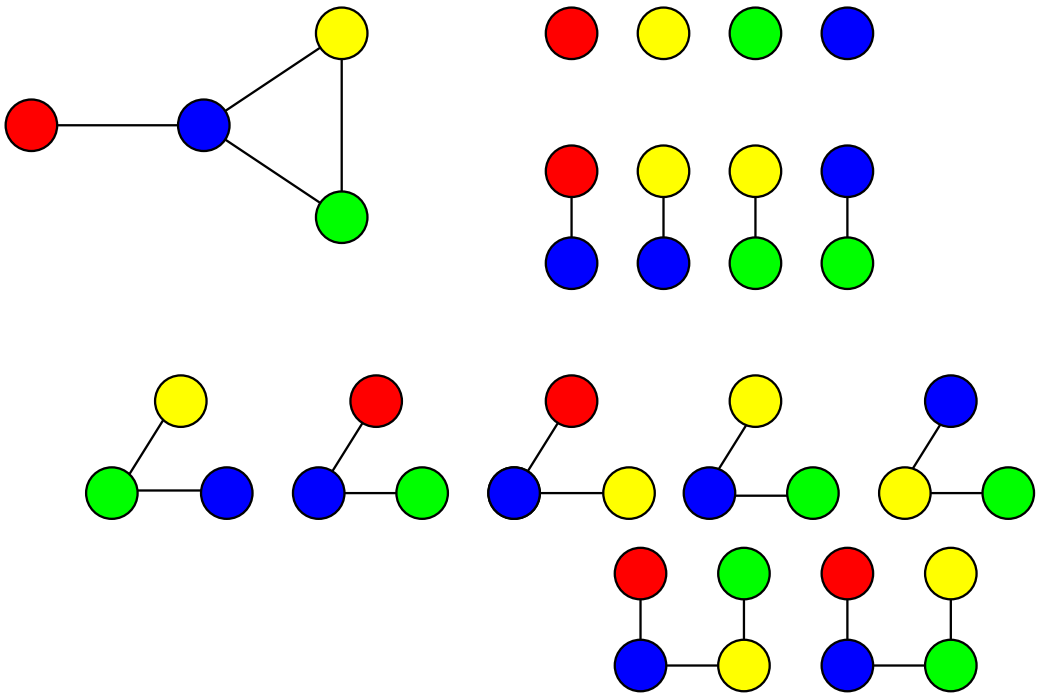
- The linear graph of size n is a subgraph of a graph X with n vertices iff X has an Hamiltonian path
- The decision problem whether a graph has a Hamiltonian path is NP-complete.

□

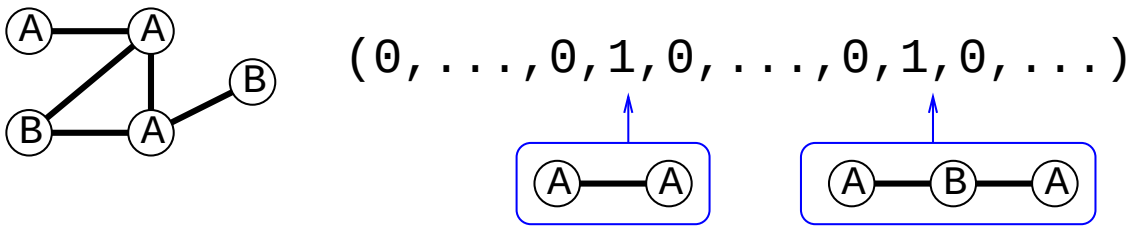
Paths

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.



Indexing by all paths?



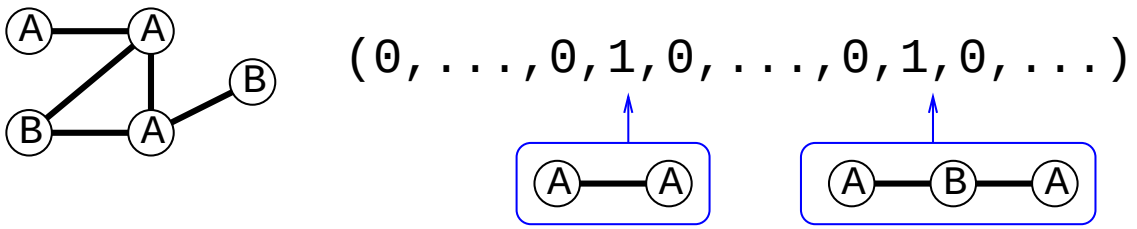
Theorem

Computing all path occurrences is NP-hard.

Proof.

Same as for subgraphs. □

Indexing by all paths?



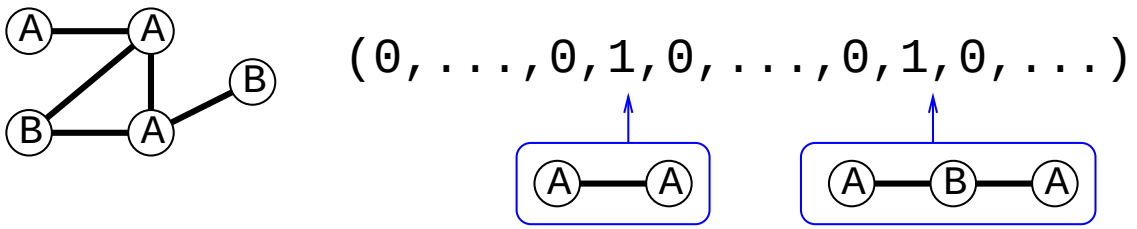
Theorem

Computing all path occurrences is NP-hard.

Proof.

Same as for subgraphs. □

Indexing by all paths?



Theorem

Computing all path occurrences is NP-hard.

Proof.

Same as for subgraphs. □

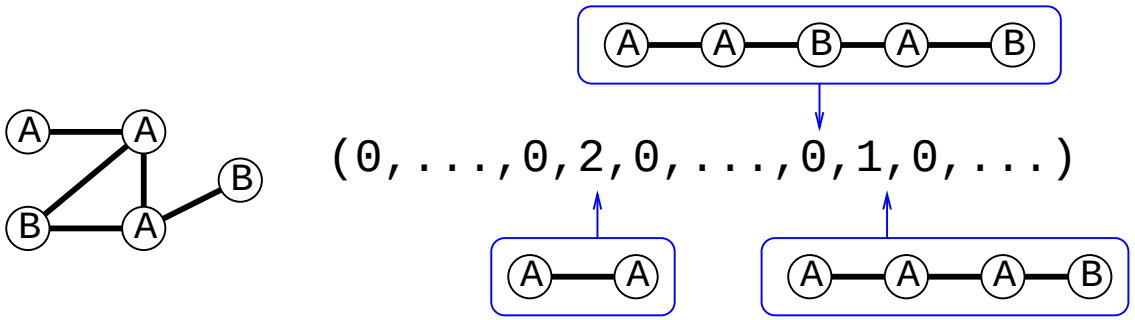
Indexing by what?

Substructure selection

We can imagine more limited sets of substuctures that lead to more computationnally efficient indexing (non-exhaustive list)

- substructures selected by domain knowledge (MDL fingerprint)
- all path up to length k (Openeye fingerprint, Nicholls 2005)
- all shortest paths (Borgwardt and Kriegel, 2005)
- all subgraphs up to k vertices (graphlet kernel, Sherashidze et al., 2009)
- all frequent subgraphs in the database (Helma et al., 2004)

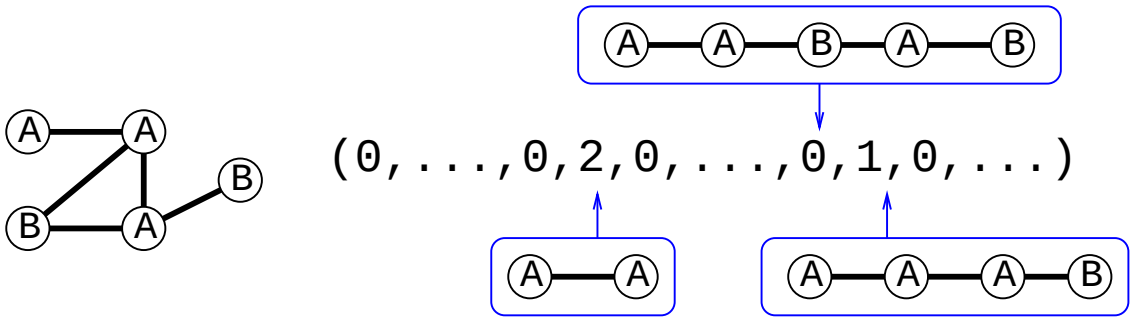
Example : Indexing by all shortest paths



Properties (Borgwardt and Kriegel, 2005)

- There are $O(n^2)$ shortest paths.
- The vector of counts can be computed in $O(n^4)$ with the Floyd-Warshall algorithm.

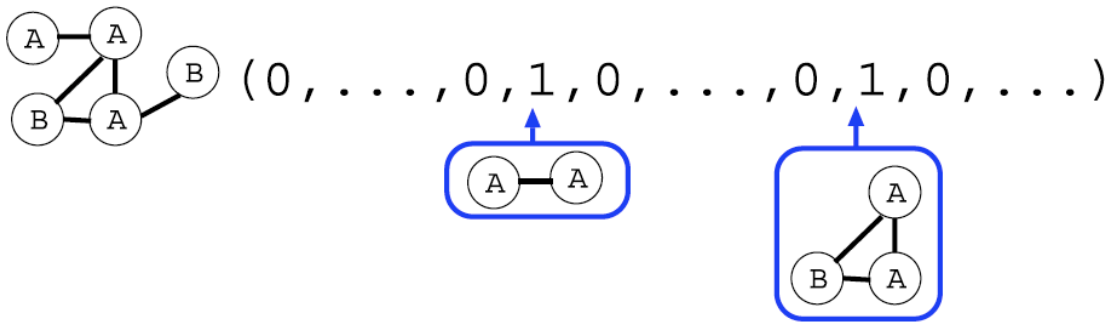
Example : Indexing by all shortest paths



Properties (Borgwardt and Kriegel, 2005)

- There are $O(n^2)$ shortest paths.
- The vector of counts can be computed in $O(n^4)$ with the Floyd-Warshall algorithm.

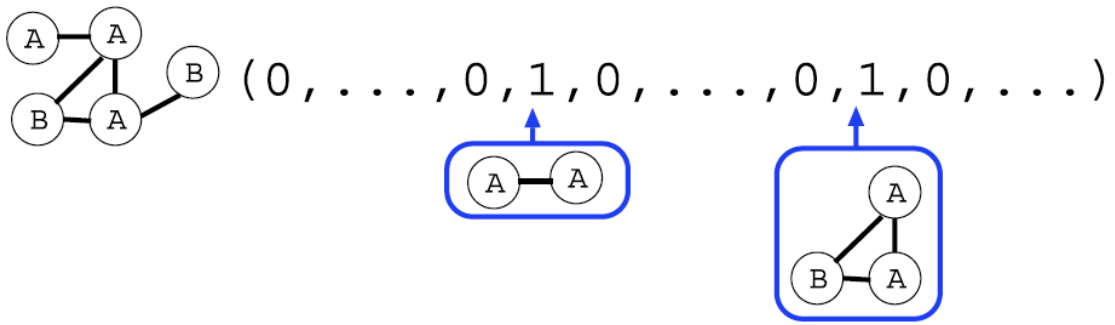
Example : Indexing by all subgraphs up to k vertices



Properties (Shervashidze et al., 2009)

- Naive enumeration scales as $O(n^k)$.
- Enumeration of connected graphlets in $O(nd^{k-1})$ for graphs with degree $\leq d$ and $k \leq 5$.
- Randomly sample subgraphs if enumeration is infeasible.

Example : Indexing by all subgraphs up to k vertices



Properties (Shervashidze et al., 2009)

- Naive enumeration scales as $O(n^k)$.
- Enumeration of connected graphlets in $O(nd^{k-1})$ for graphs with degree $\leq d$ and $k \leq 5$.
- Randomly sample subgraphs if enumeration is infeasible.

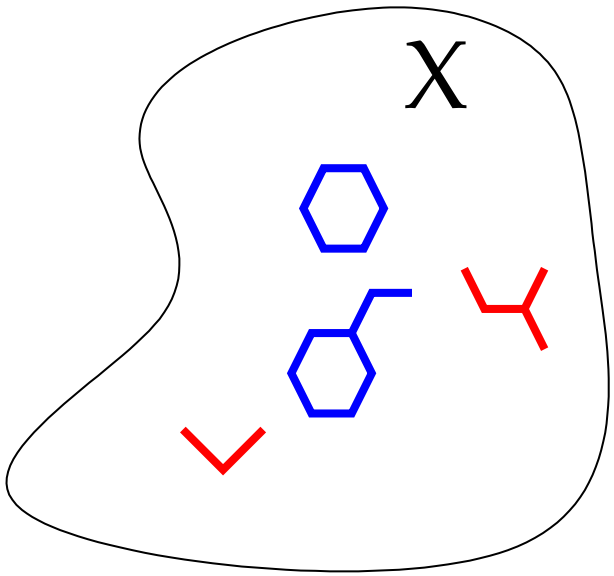
Summary

- Explicit computation of substructure occurrences can be computationnally prohibitive (subgraph, paths)
- Several ideas to reduce the set of substructures considered
- In practice, NP-hardness may not be so prohibitive (e.g., graphs with small degrees), the strategy followed should depend on the data considered.

Outline

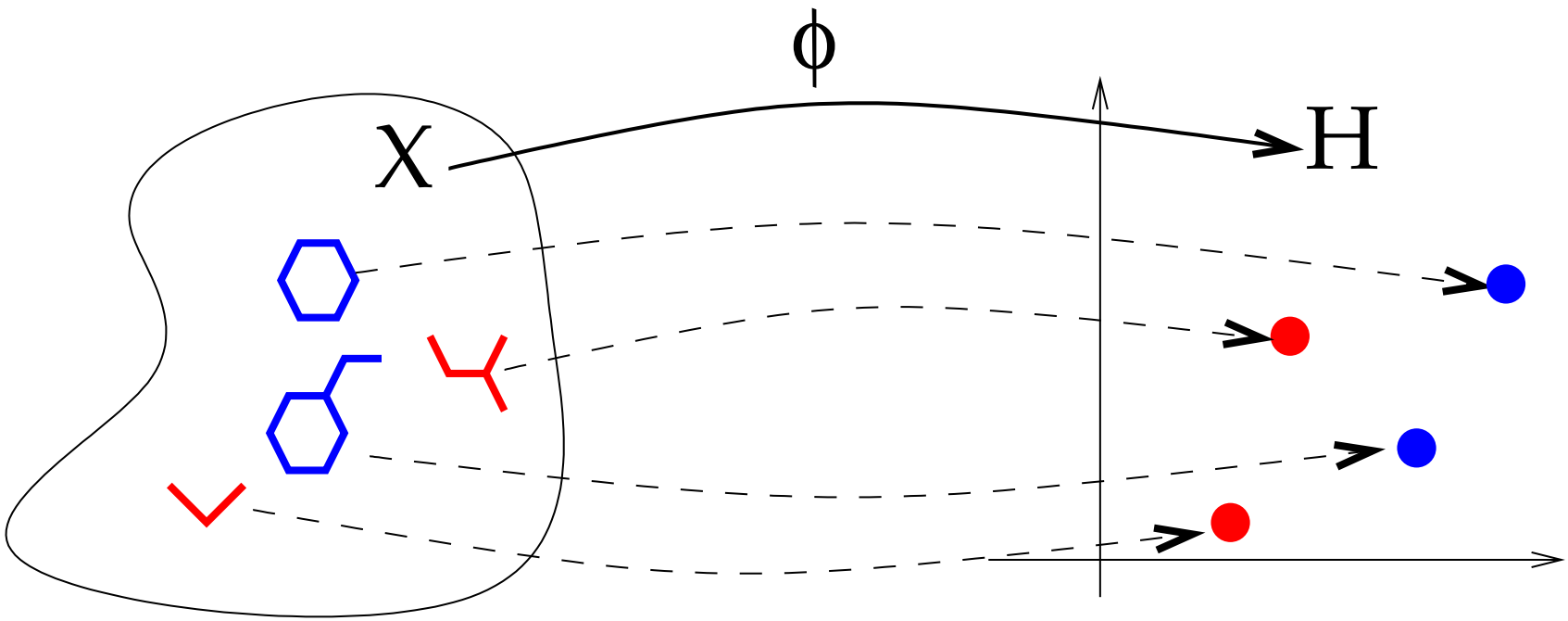
The idea

1. Represent implicitly each graph x by a vector $\Phi(x) \in \mathcal{H}$ through the kernel
$$K(x, x') = \Phi(x)^\top \Phi(x') \,.$$
2. Use a kernel method for classification in $\mathcal{H}$.



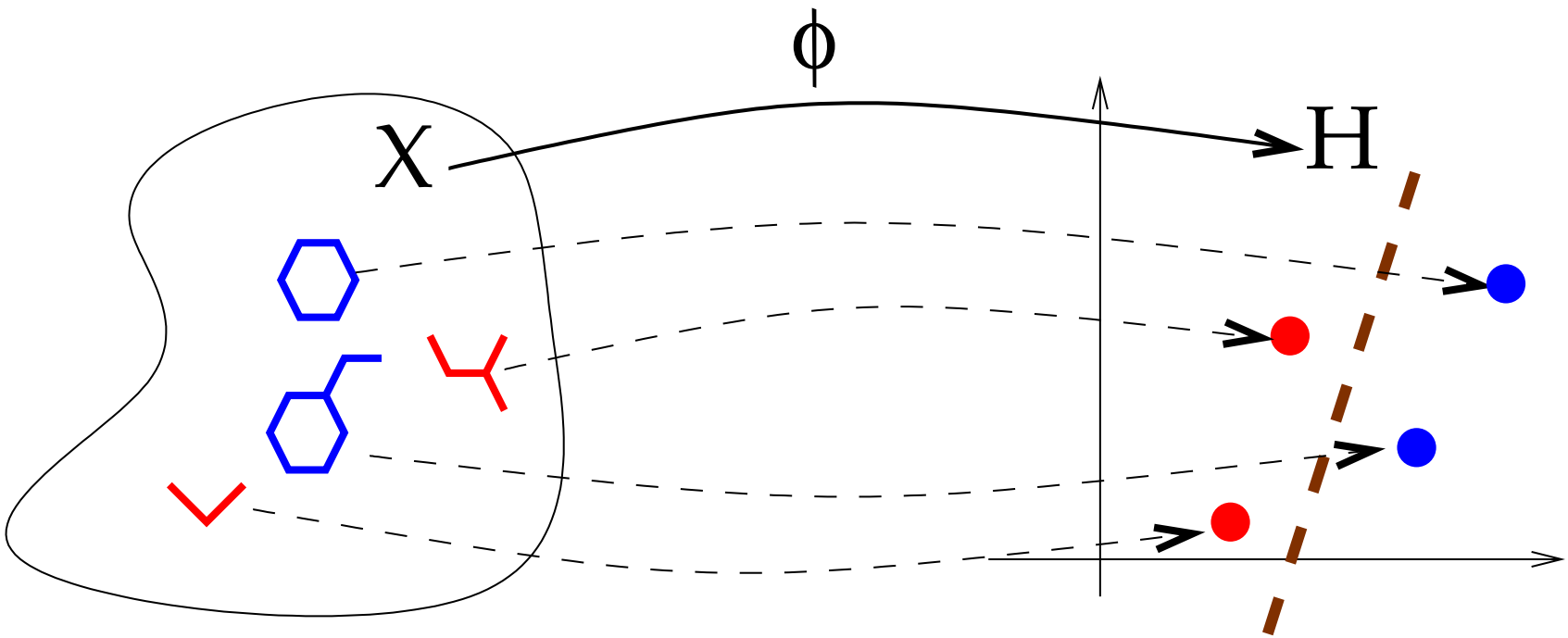
The idea

1. Represent implicitly each graph x by a vector $\Phi(x) \in \mathcal{H}$ through the kernel
$$K(x, x') = \Phi(x)^\top \Phi(x') .$$
2. Use a kernel method for classification in $\mathcal{H}$.



The idea

1. Represent implicitly each graph x by a vector $\Phi(x) \in \mathcal{H}$ through the kernel
$$K(x, x') = \Phi(x)^\top \Phi(x') .$$
2. Use a kernel method for classification in $\mathcal{H}$.



Expressiveness vs Complexity

Definition: Complete graph kernels

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

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

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

Expressiveness vs Complexity trade-off

- If a graph kernel is not complete, then there is no hope to learn all possible functions over $\mathcal{X}$: 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.:

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

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

Expressiveness vs Complexity trade-off

- If a graph kernel is not complete, then there is no hope to learn all possible functions over $\mathcal{X}$: 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 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 $G_1 \simeq G_2$). □

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 $G_1 \simeq G_2$). $\square$

Subgraph kernel

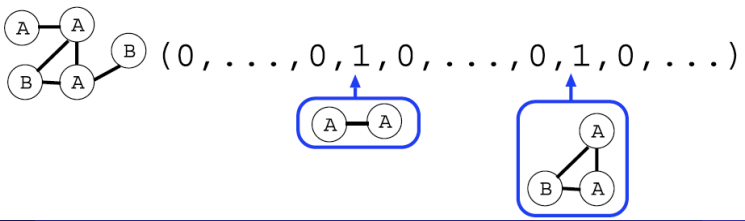
Definition

- Let $(\lambda_G)_{G \in \mathcal{X}}$ a set or nonnegative real-valued weights
- For any graph $G \in \mathcal{X}$, 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 $G_2 \in \mathcal{X}$ 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.

Proof (1/2)

- Let P_n be the path graph with n vertices.
- Subgraphs of P_n are path graphs:

$$\Phi(P_n) = n e_{P_1} + (n-1) e_{P_2} + \ldots + e_{P_n} .$$

- The vectors $\Phi(P_1), \ldots, \Phi(P_n)$ are linearly independent, therefore:

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

where 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 vertices.
- Subgraphs of P_n are path graphs:

$$\Phi(P_n) = n e_{P_1} + (n-1) e_{P_2} + \ldots + e_{P_n} .$$

- The vectors $\Phi(P_1), \ldots, \Phi(P_n)$ are linearly independent, therefore:

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

where 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.

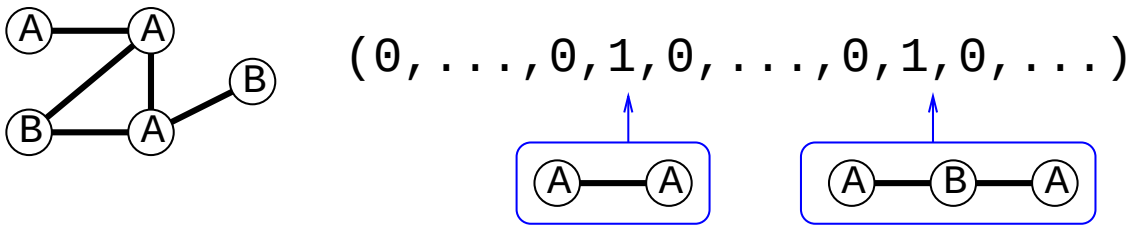
Proof (2/2)

- 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.,

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

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

Path kernel



Definition

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

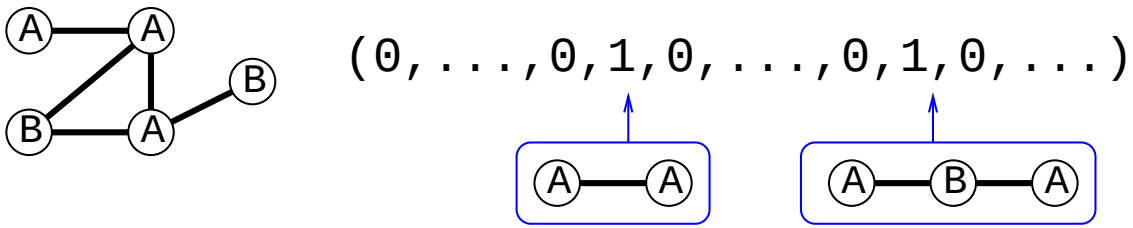
$$K_{path}(G_1, G_2) = \sum_{H \in \mathcal{P}} \lambda_H \Phi_H(G_1) \Phi_H(G_2) \,,$$

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

Proposition (Gärtner et al., 2003)

Computing the path kernel is NP-hard.

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

Proposition (Gärtner et al., 2003)

Computing the path kernel is NP-hard.

Summary

Expressiveness vs Complexity trade-off

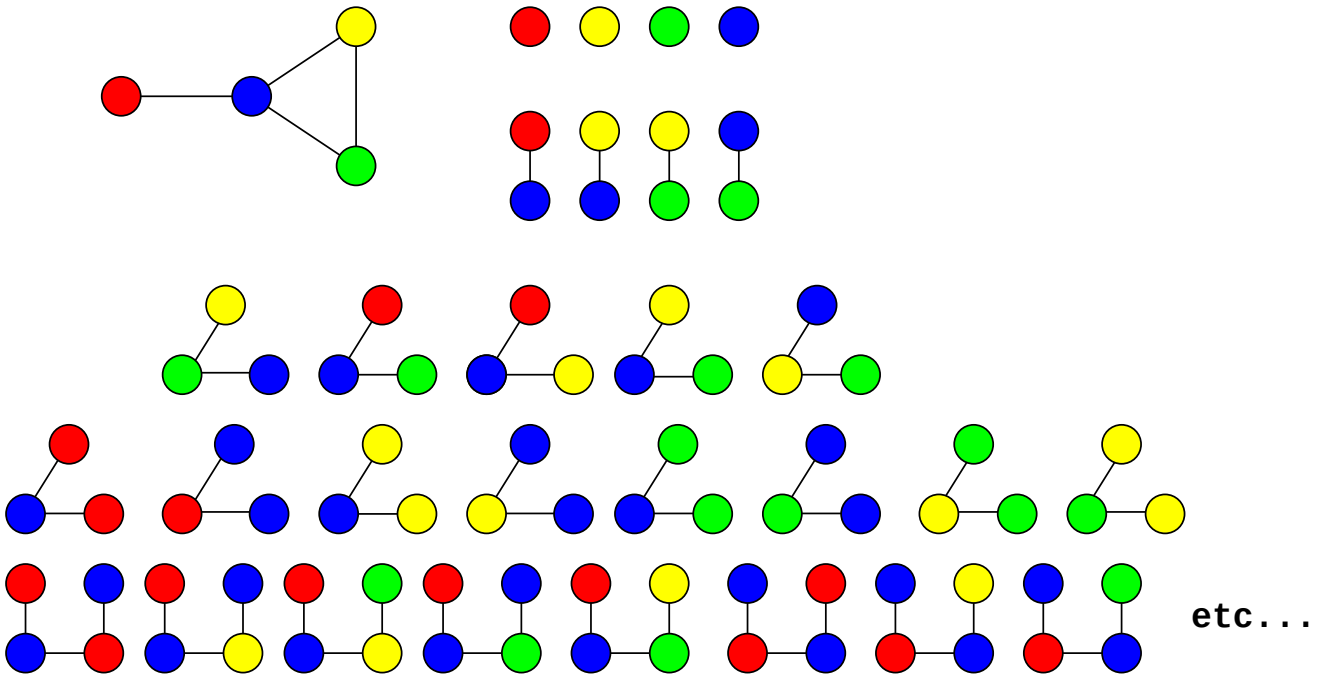
- It is intractable to compute complete graph kernels.
- It is intractable to compute the subgraph kernels.
- Restricting subgraphs to be linear does not help: it is also intractable to compute the path kernel.
- One approach to define polynomial time computable graph kernels is to have the feature space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

Outline

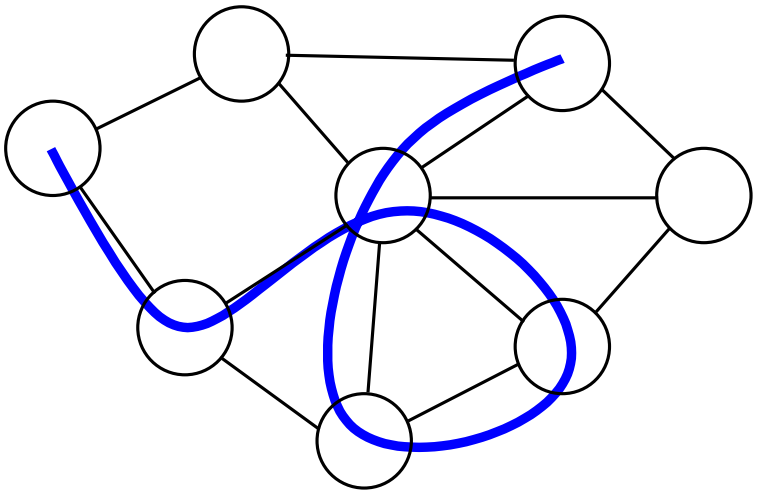
Walks

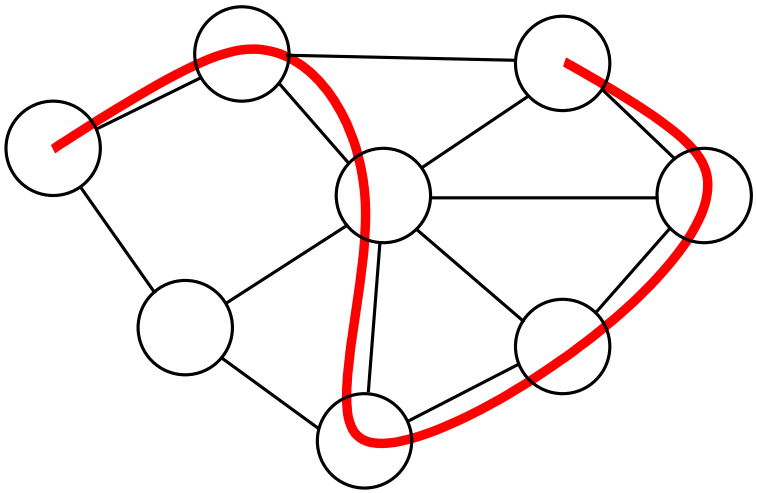
Definition

- 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.



Walks $\neq$ paths





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

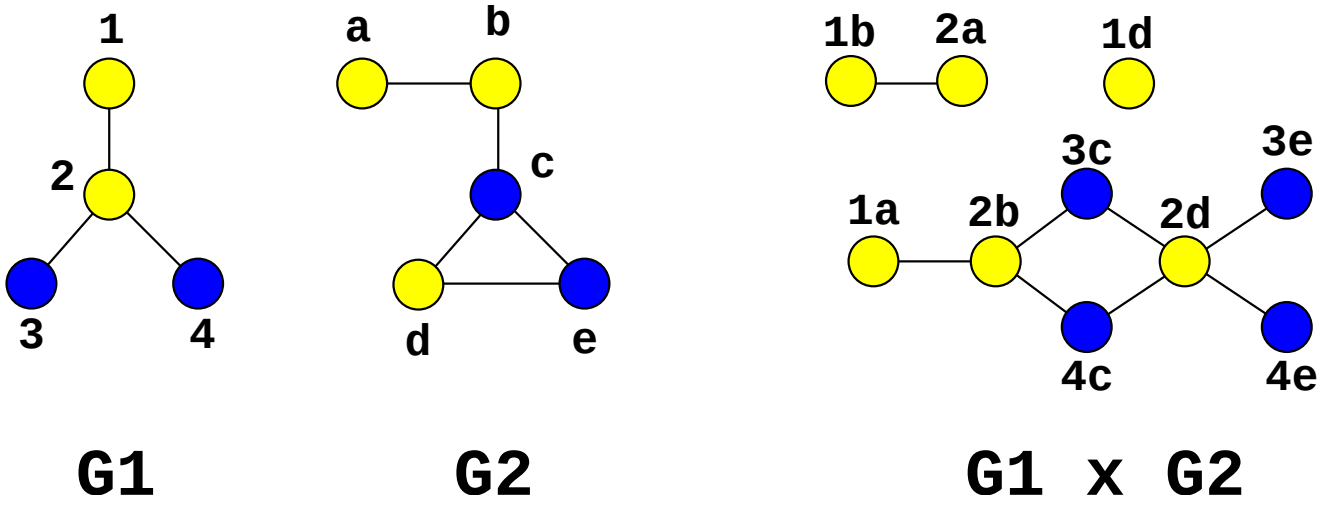
These three kernels (*n*th-order, random and geometric walk kernels) can be computed efficiently in polynomial time.

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

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

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:

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

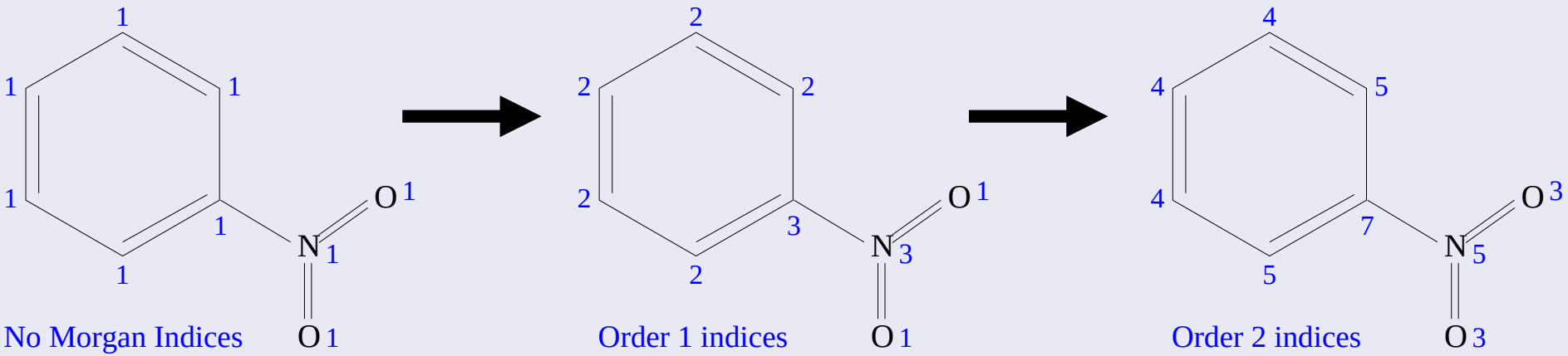
- Let Λ_i be the vector of $\lambda^i(v)$ and Λ_t be the matrix of $\lambda^t(v, v')$:

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

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

Extensions 1: label enrichment

Atom relabebling with the Morgan index

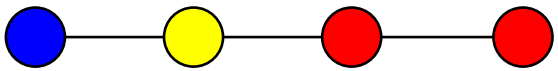


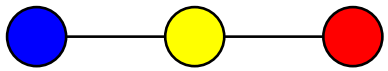
- Compromise between fingerprints and structural keys features.
- Other relabeling schemes are possible (graph coloring).
- Faster computation with more labels (less matches implies a smaller product graph).

Extension 2: Non-tottering walk kernel

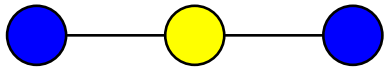
Tottering walks

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





Non-tottering

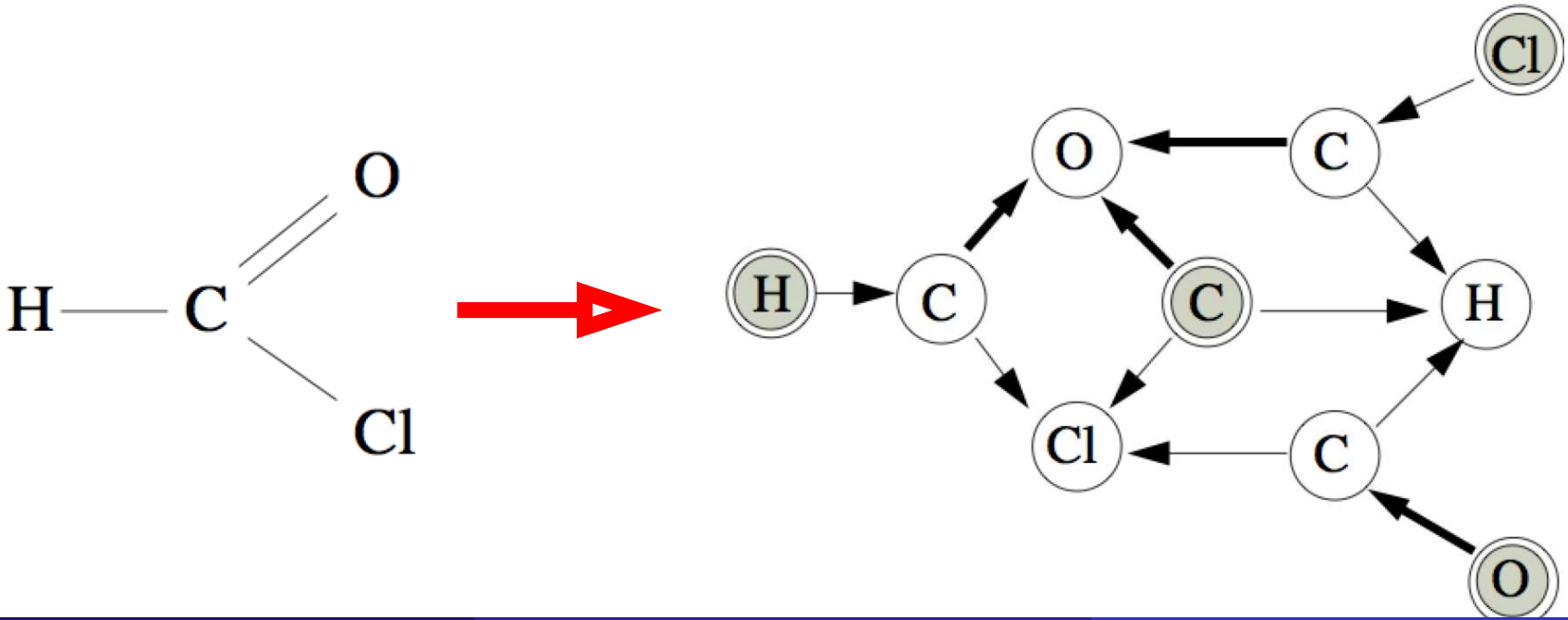


Tottering

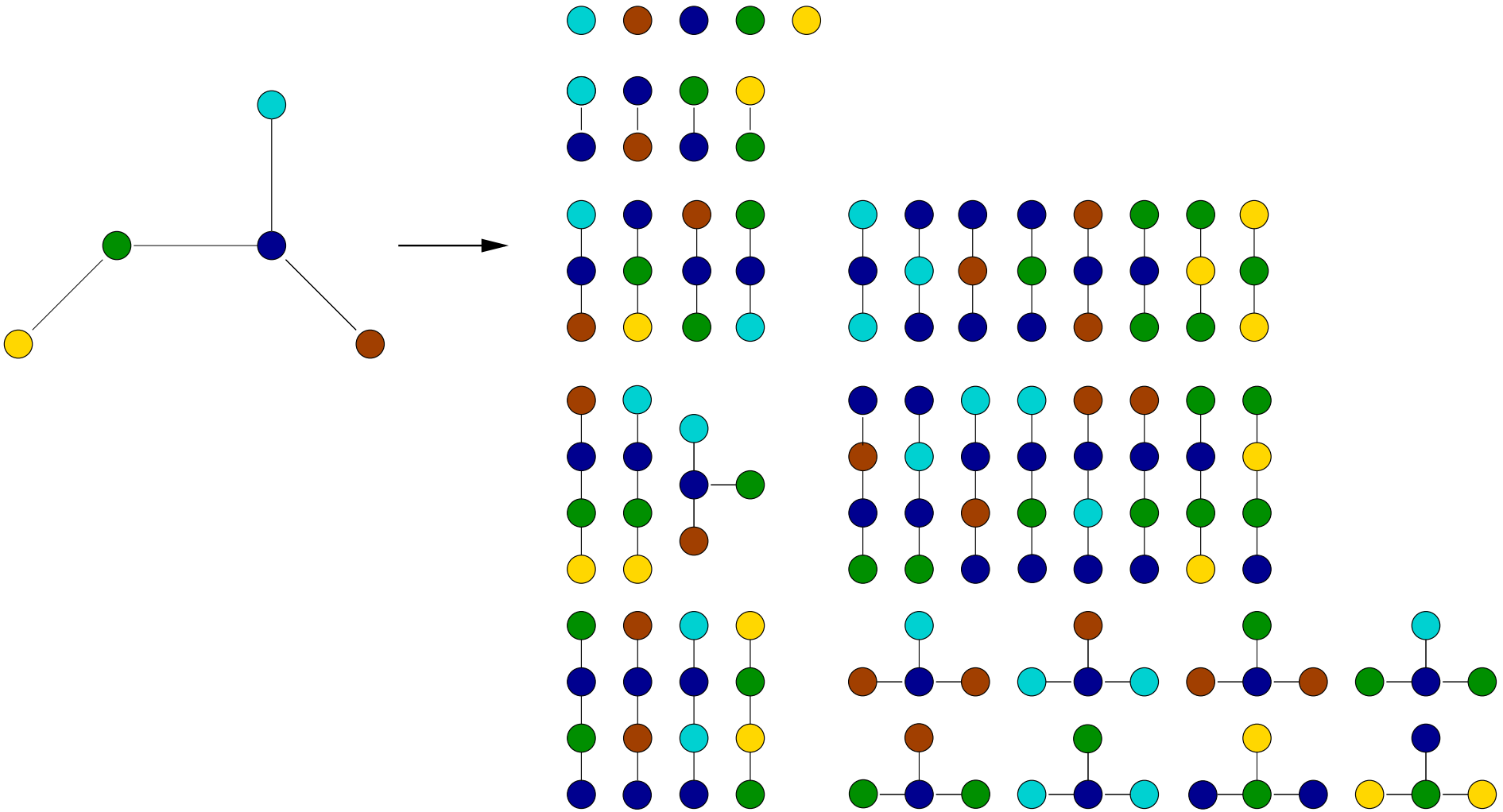
- Tottering walks seem irrelevant for many applications
- Focusing on non-tottering walks is a way to get closer to the path kernel (e.g., equivalent on trees).

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

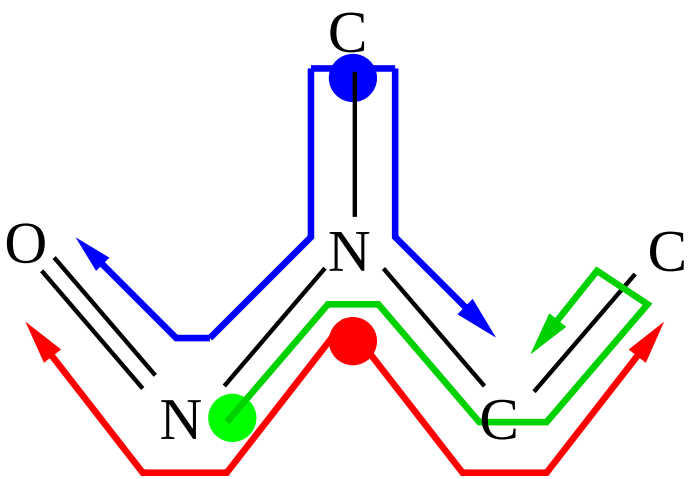
- Second-order Markov random walk to prevent tottering walks
- Written as a first-order Markov random walk on an augmented graph
- Normal walk kernel on the augmented graph (which is always a directed graph).

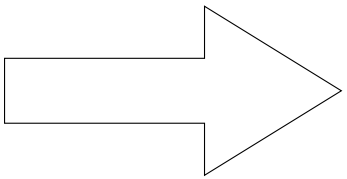


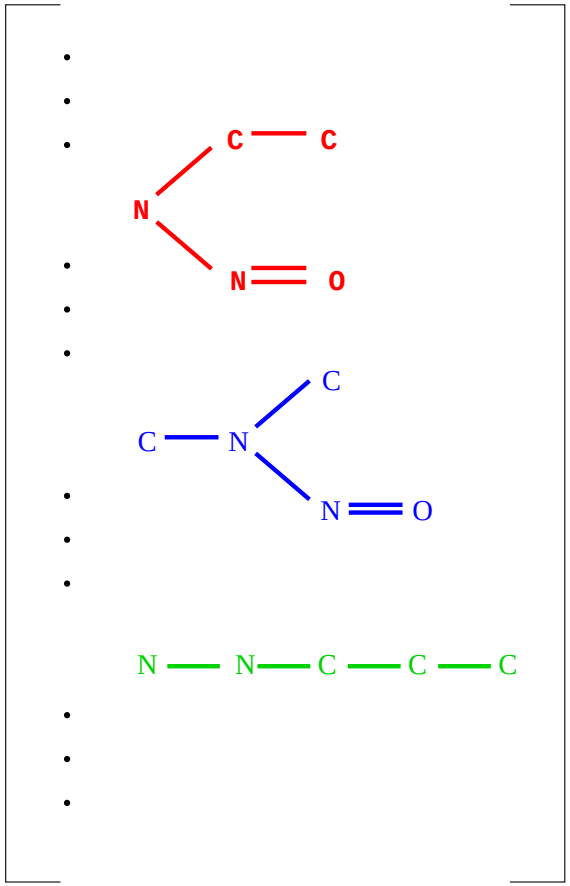
Extension 3: Subtree kernels



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.

Outline

Application in chemoinformatics (Mahé et al., 2004)

MUTAG dataset

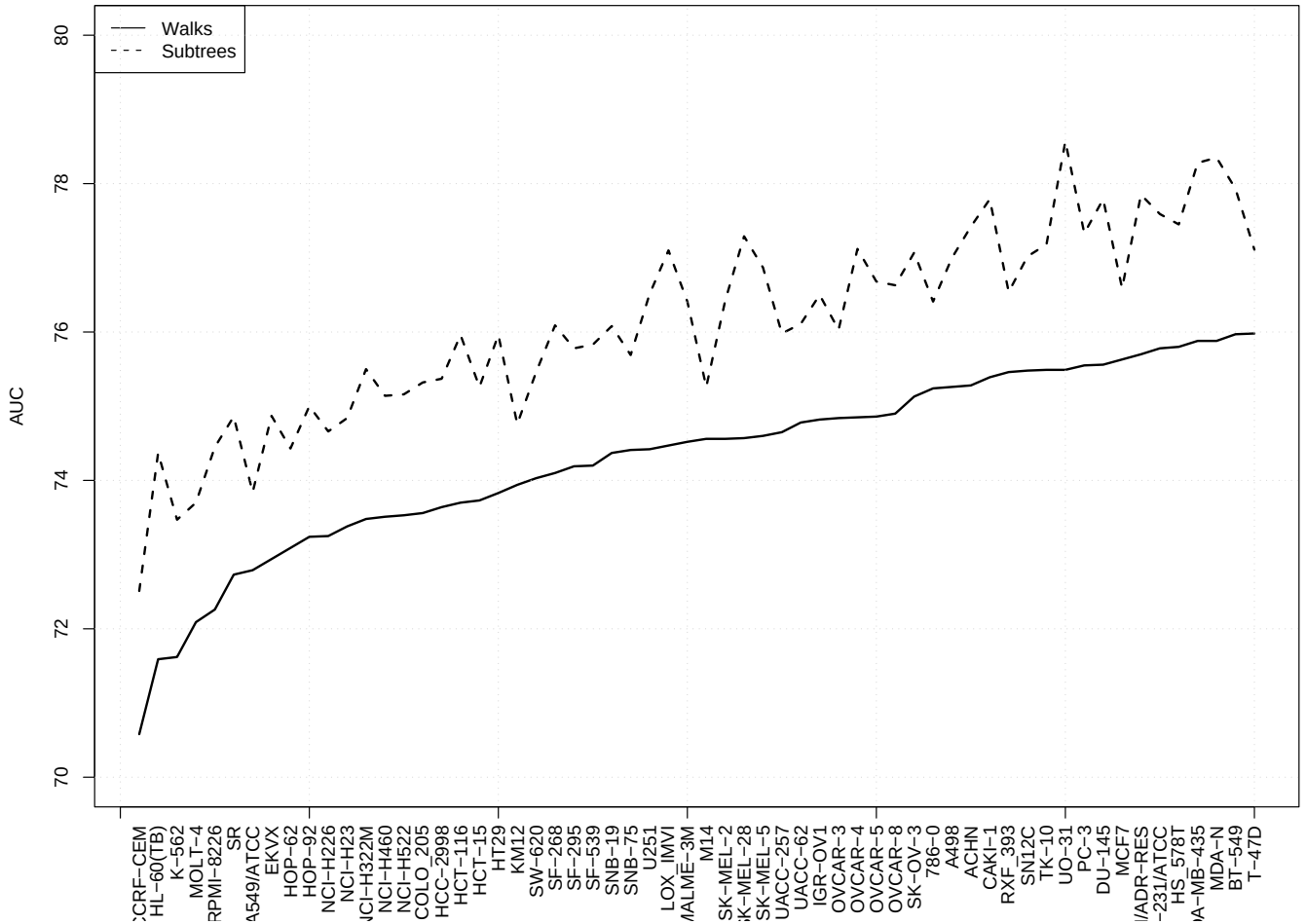
- aromatic/hetero-aromatic compounds
- high mutagenic activity /no mutagenic activity, assayed in *Salmonella typhimurium*.
- 188 compouunds: 125 + / 63 -

Results

10-fold cross-validation accuracy

Method	Accuracy
Progol1	81.4%
2D kernel	91.2%

2D Subtree vs walk kernels

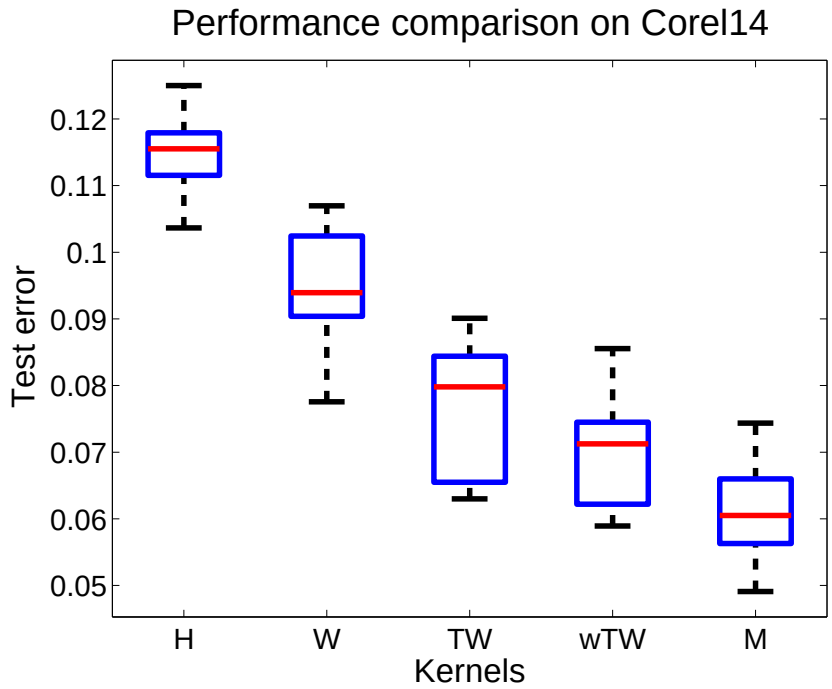


Screening of inhibitors for 60 cancer cell lines.

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).



Summary: graph kernels

What we saw

- Kernels do not allow to overcome the NP-hardness of subgraph patterns
- They allow to work with approximate subgraphs (walks, subtrees), in infinite dimension, thanks to the kernel trick
- However: using kernels makes it difficult to come back to patterns after the learning stage

Outline

Outline

Gene expression

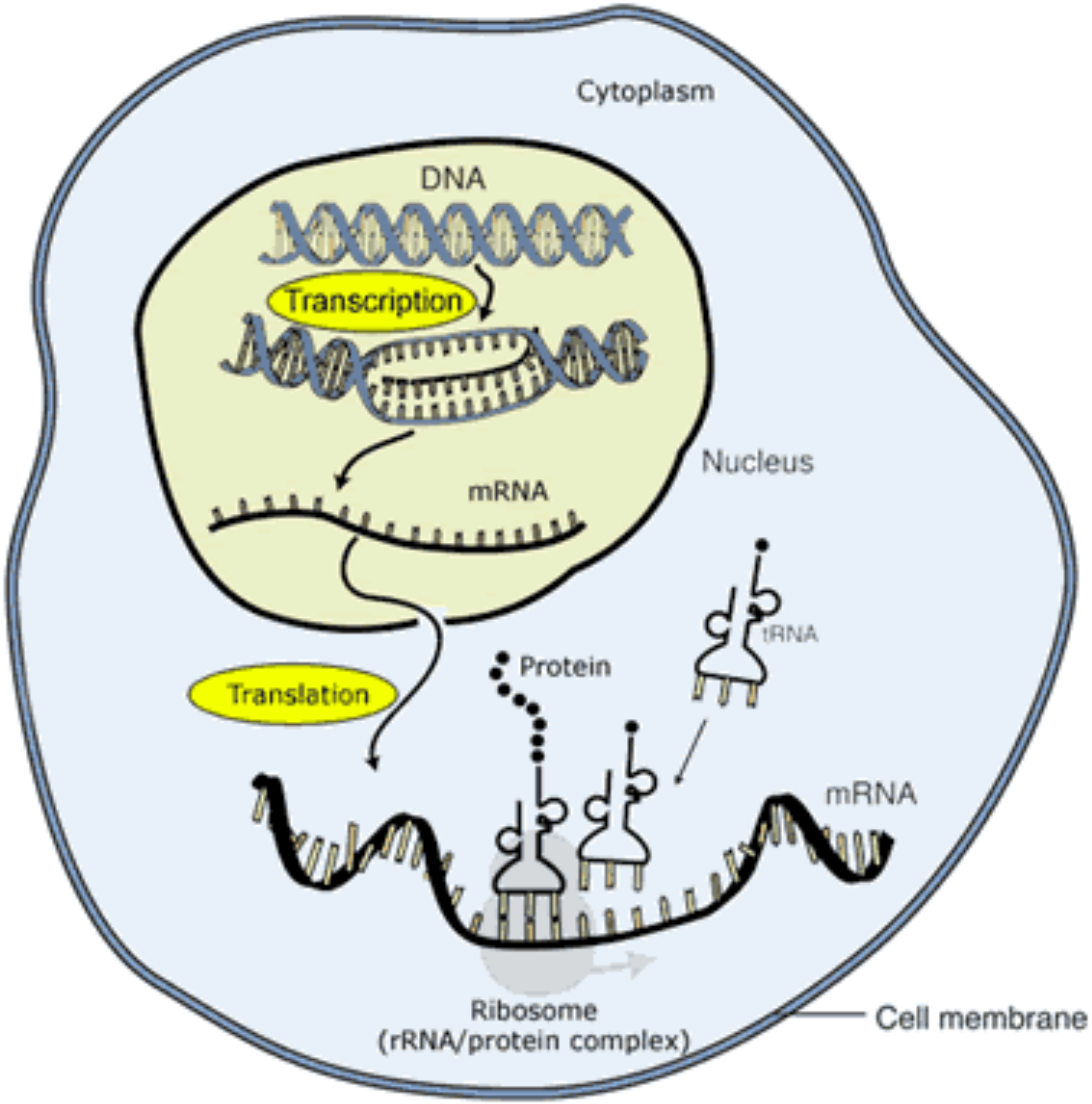
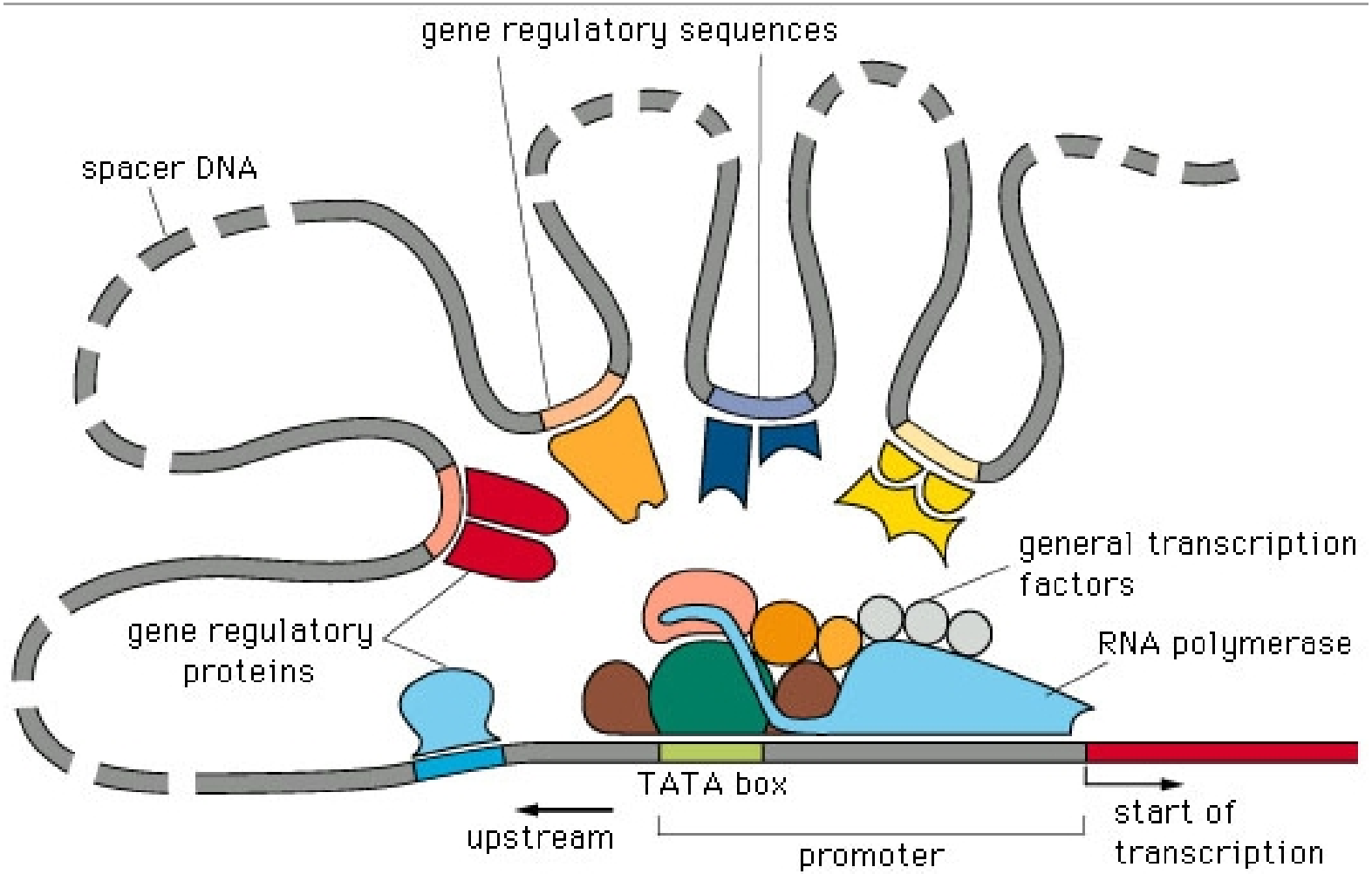
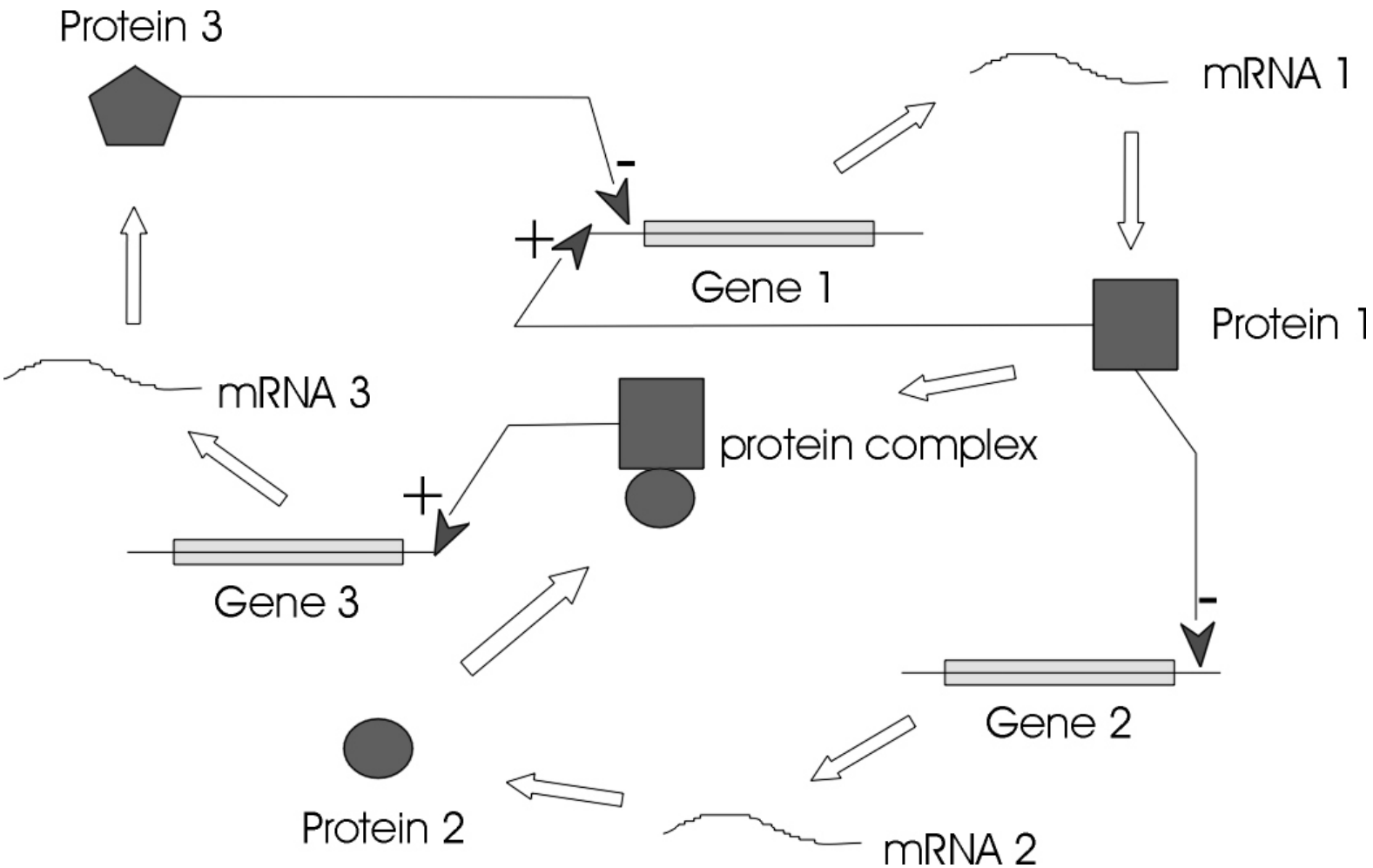


Image adapted from: National Human Genome Research Institute.

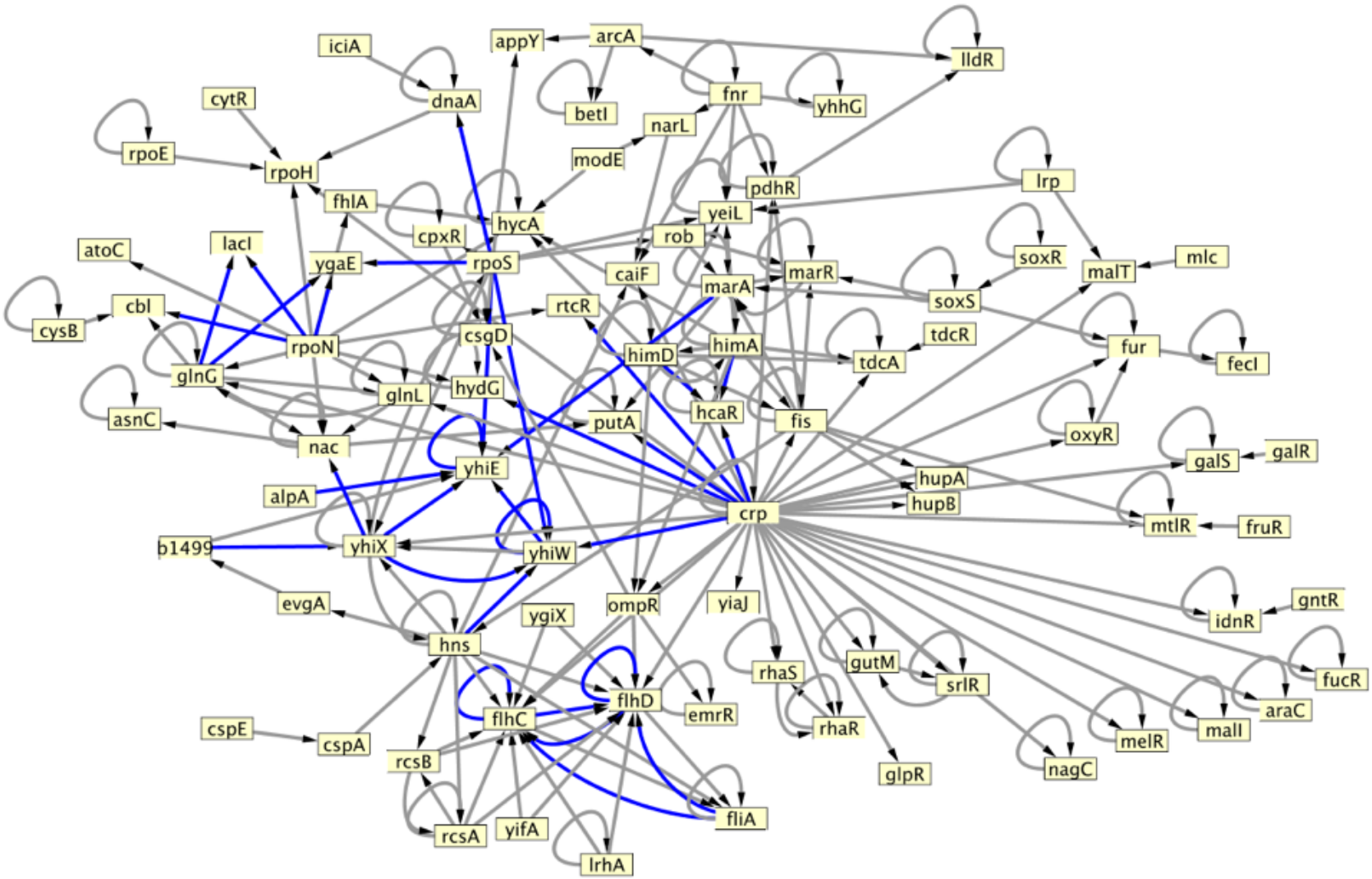
Gene expression regulation



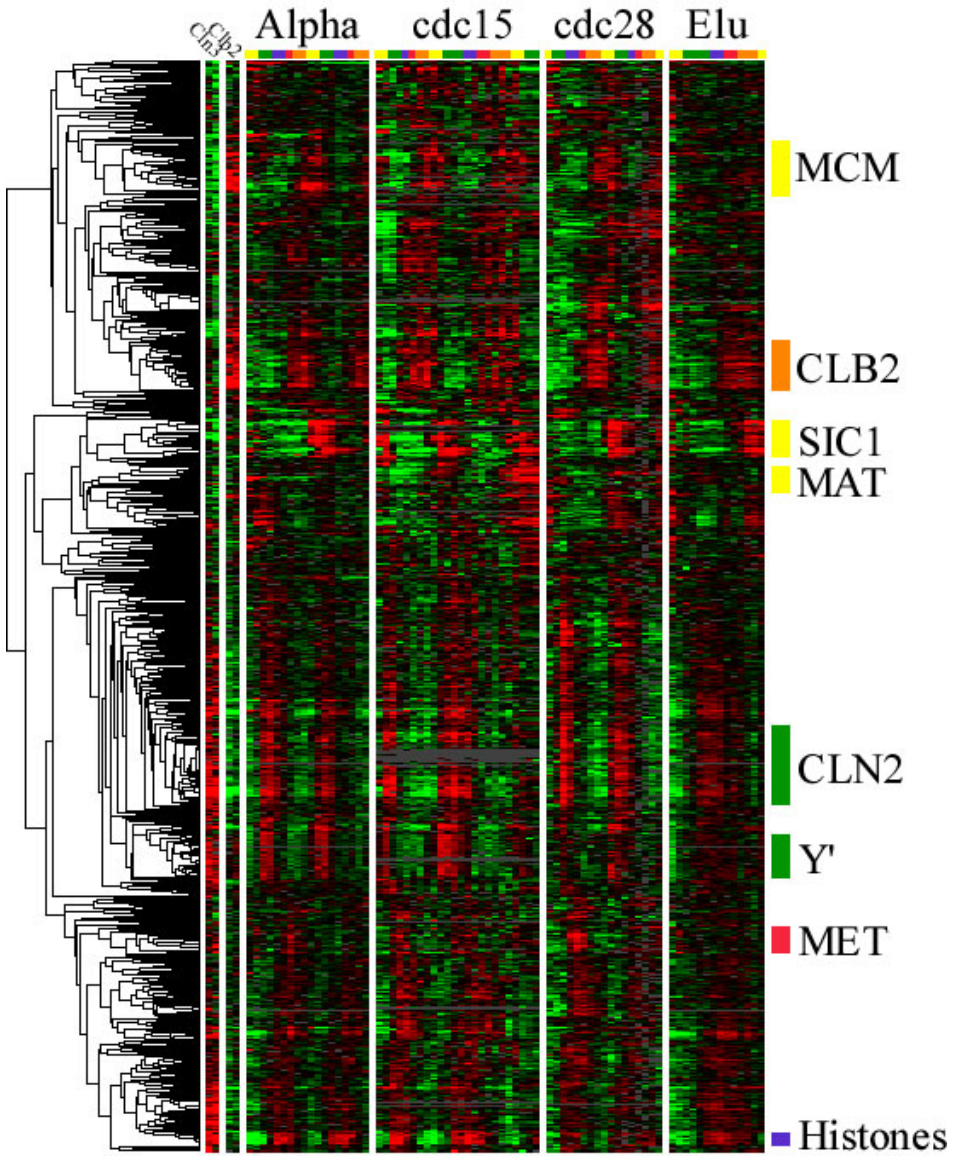
Gene regulatory network



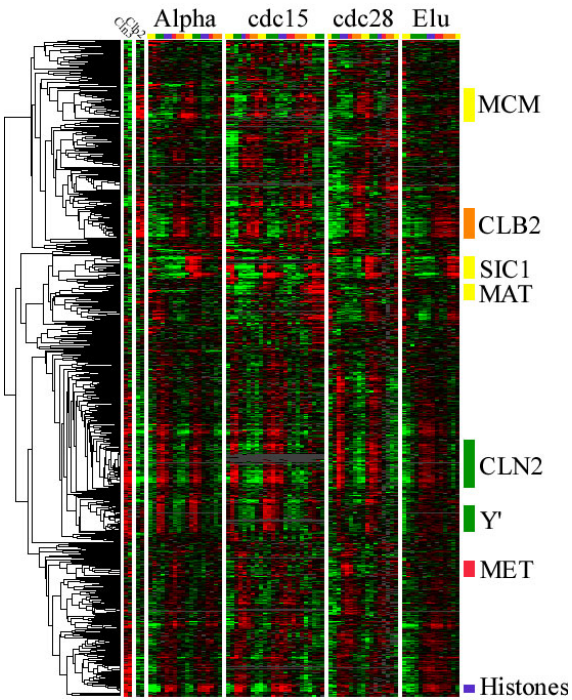
Gene regulatory network of E. coli

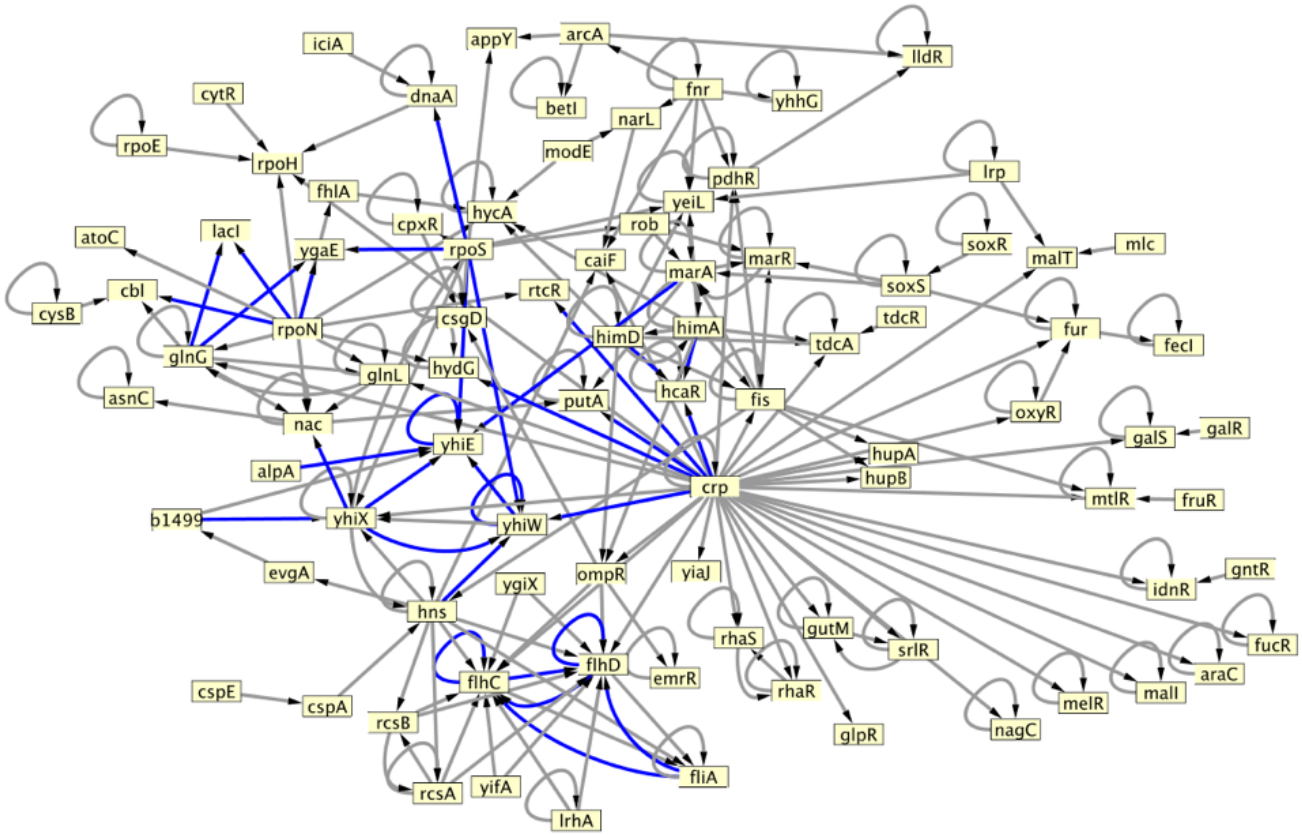


Gene expression data

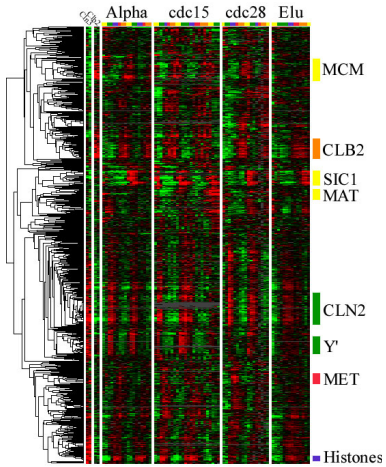


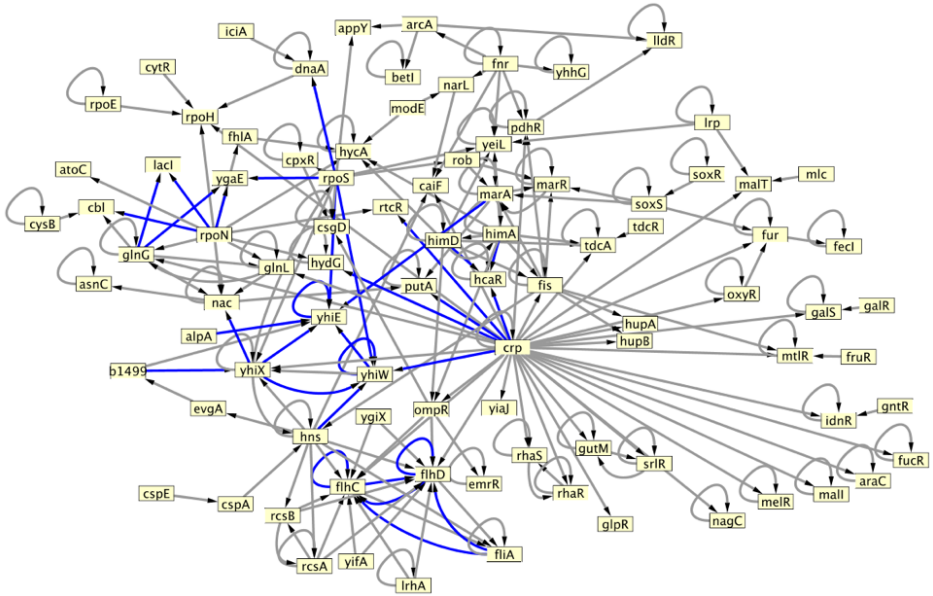
Reconstruction of gene regulatory network





Two flavours: *de novo* or supervised





De novo inference

Given a matrix of expression data, infer regulations

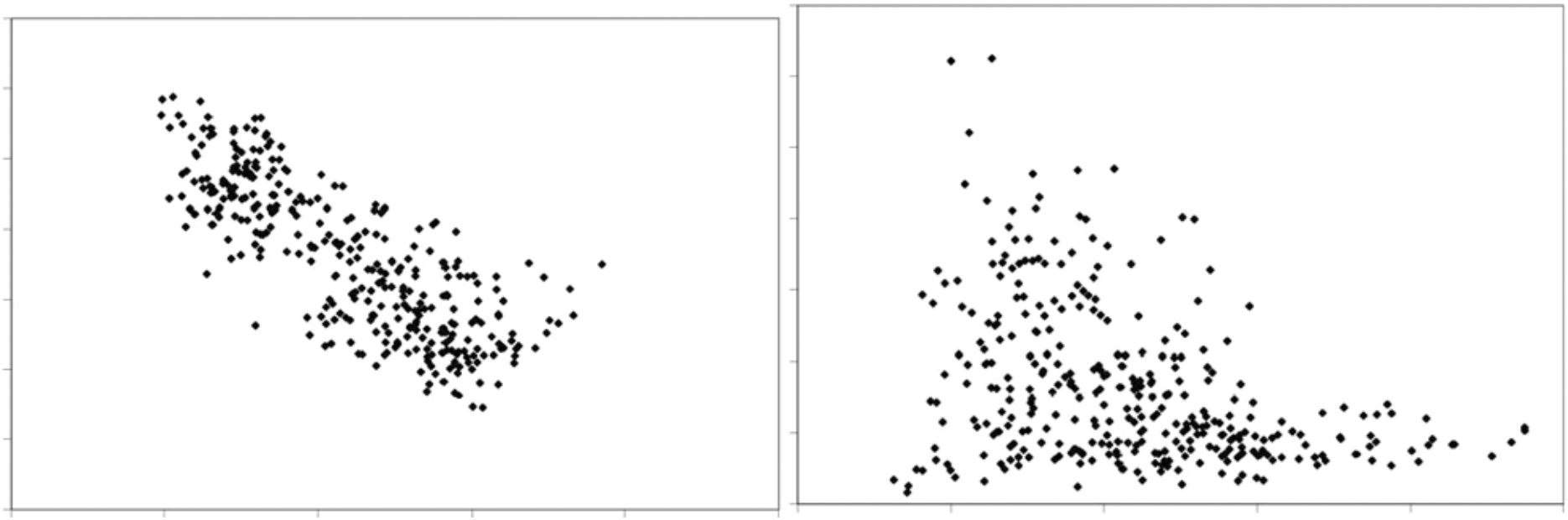
Supervised inference

Given a matrix of expression data and a set of knows regulations, infer *other unknown* regulations

Outline

The idea

If A regulates B, then we should expect some form of "correlation" between the expression levels of A and B across different experiments.



We can therefore try to detect these correlations to infer regulation.

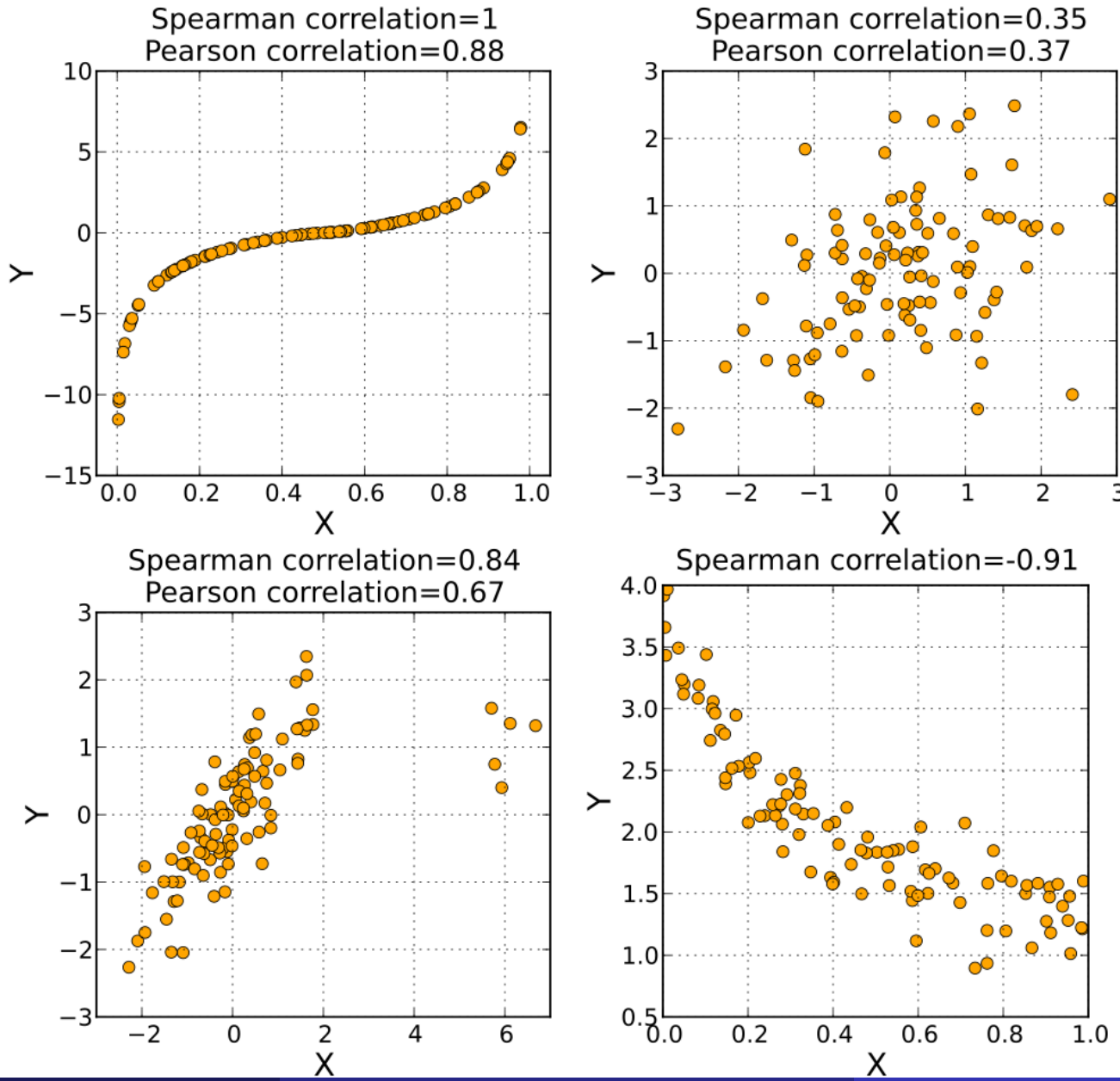
Measuring dependency: correlation coefficients

- $(X_1, Y_1), \ldots, (X_n, Y_n)$ the n expression values of both genes
- Pearson correlation:

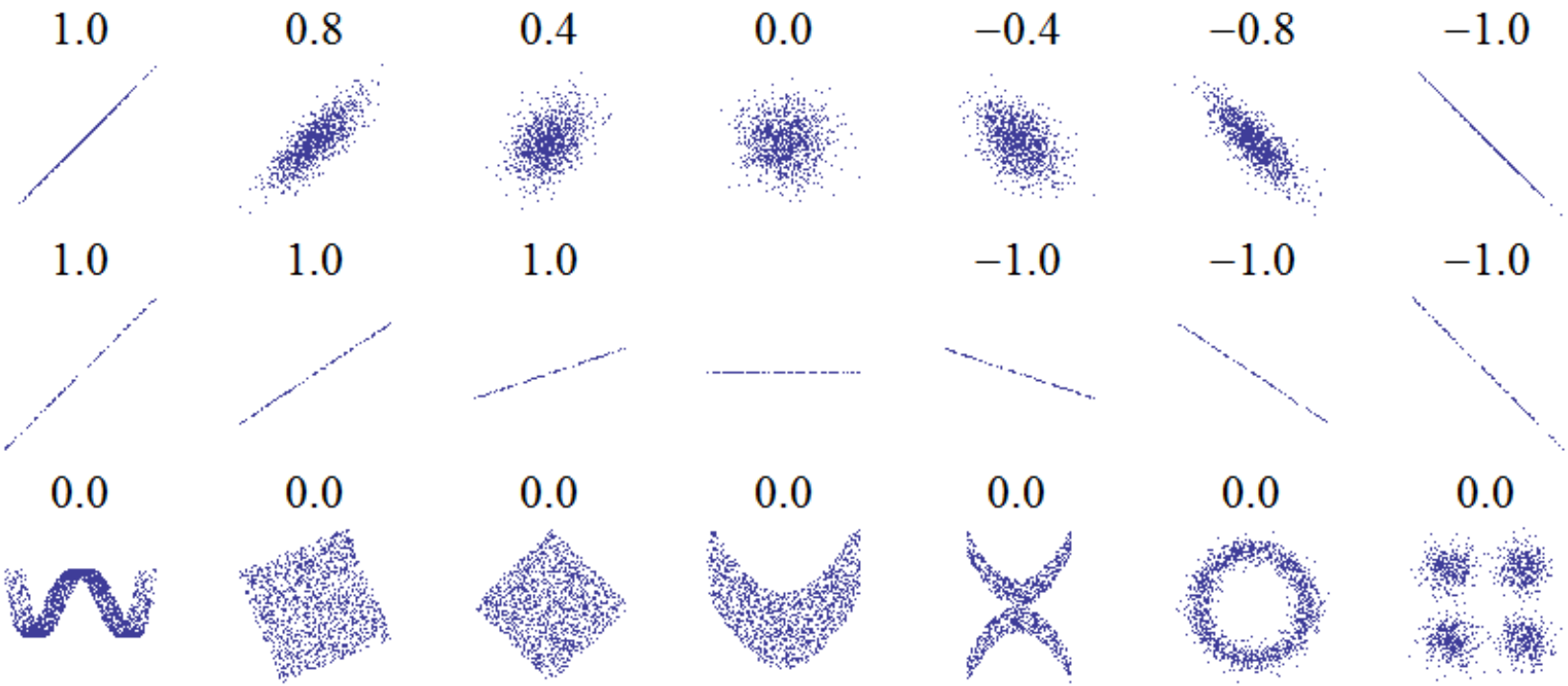
$$\rho = \frac{cov(X, Y)}{\sigma_X \sigma_Y} = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i (X_i - \bar{X})^2} \sqrt{\sum_i (Y_i - \bar{Y})^2}}$$

- Spearman correlation: similar but replace X_i by its rank.

Illustration



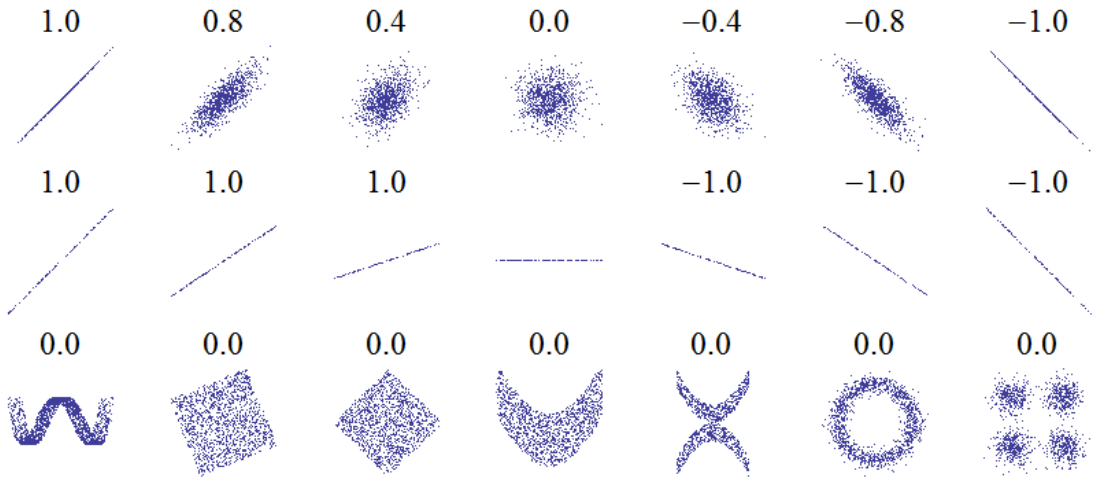
Limit of correlations



Mutual information

$$I(X;Y) = \int_Y \int_X p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) dxdy$$

- $I(X;Y) \geq 0$
- $I(X;Y) = 0$ if and only if X and Y are independent



Outline

The idea

- The dynamic equation of the mRNA concentration of a gene is of the form:

$$\frac{dX}{dt} = f(X, R)$$

  where R represent the set of concentrations of transcription factors that regulate X.
- At steady state, $dX/dt = 0 = f(X, R)$
- If we linearize $f(X, R) = 0$ we get linear relation of the form

$$X = \sum_{i \in R} \beta_i X_i$$

- This suggests to look for sets of transcription factors whose concentration is sufficient to explain the level of X across different experiments.

Predicting regulation by sparse regression

Let Y the expression of a gene, and $X_1, \ldots, X_p$ the expression of all TFs. We look for a model

$$Y = \sum_{i=1}^{p} \beta_i X_i + \text{noise}$$

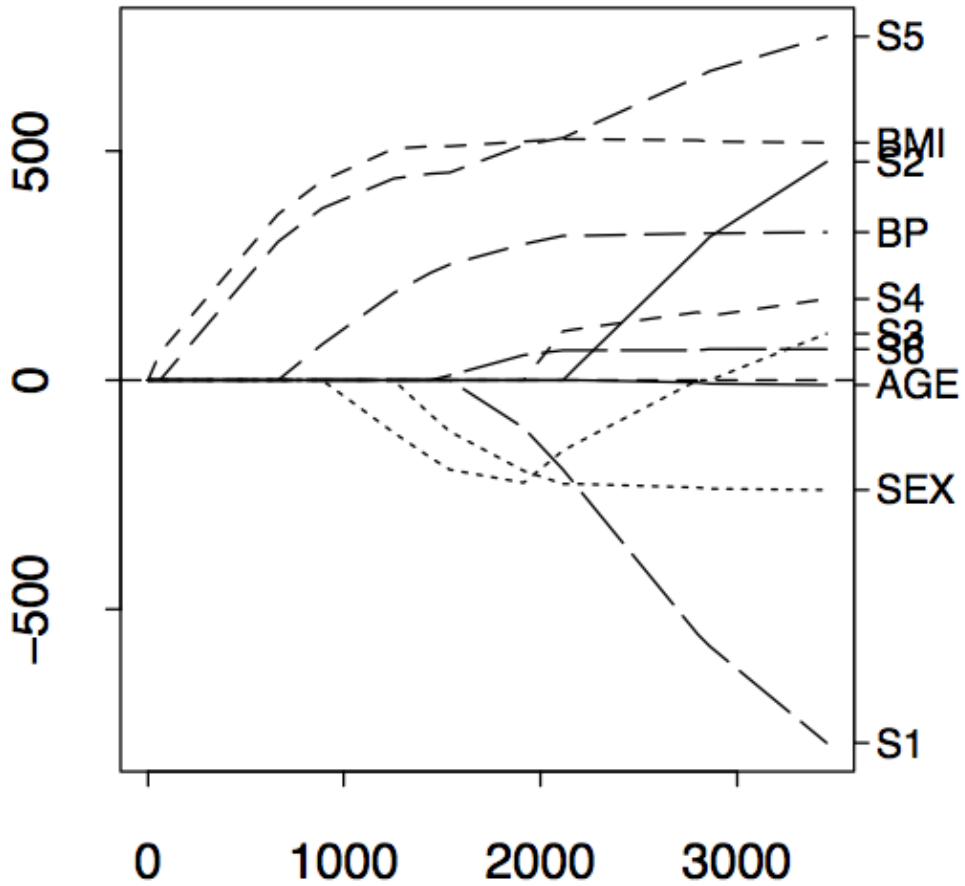
where β is sparse, i.e., only a few β_i are non-zero.
We can estimate the sparse regression model from a matrix of expression data.
Non-zero β_i's correspond to predicted regulators.

Example: sparse regression with the Lasso

$$\min_{\beta \in \mathbb{R}^p} \sum i = 1^n \left(Y_i - \sum_{j=1}^{p} X_i, j\beta_j \right)^2 \text{ such that } \sum_{i=1}^{p} | \beta_i | \leq t$$

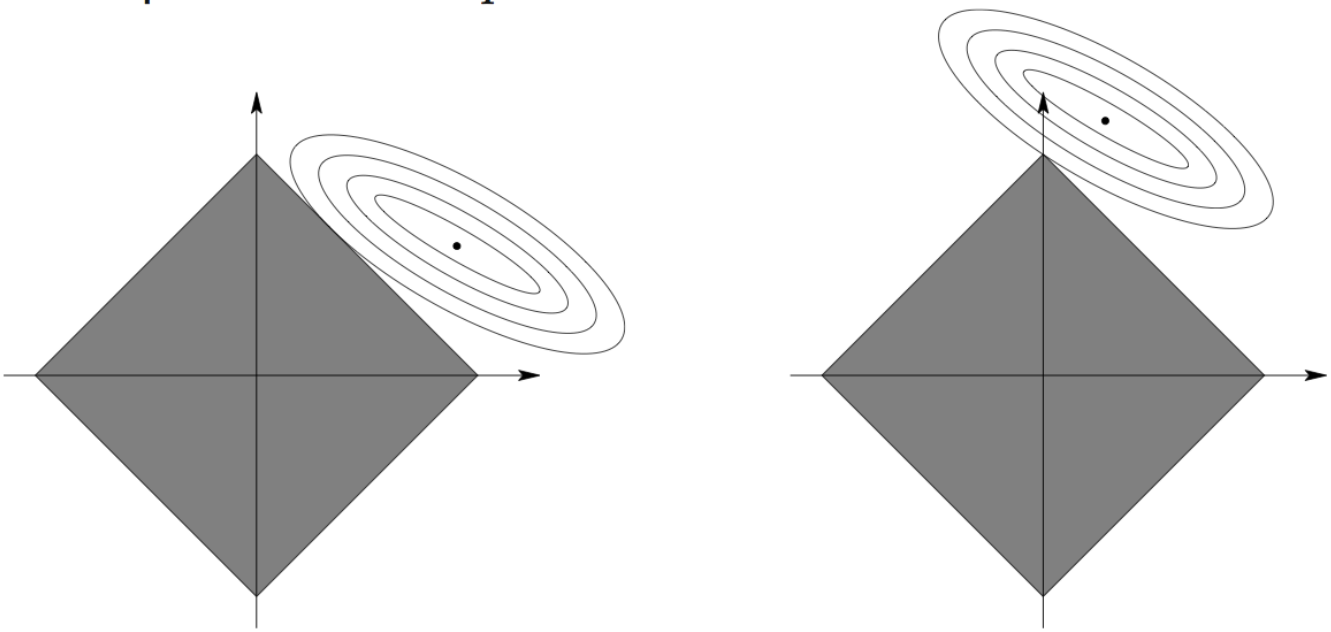
- No explicit solution, but this is just a quadratic program.
- LARS (Efron et al., 2004) provides a fast algorithm to compute the solution for all t's simultaneously (regularization path)
- When t is not too large, the solution will usually be sparse

LASSO regression example



Why LASSO leads to sparse solutions

Geometric interpretation with $p = 2$



Improved feature selection with stability selection

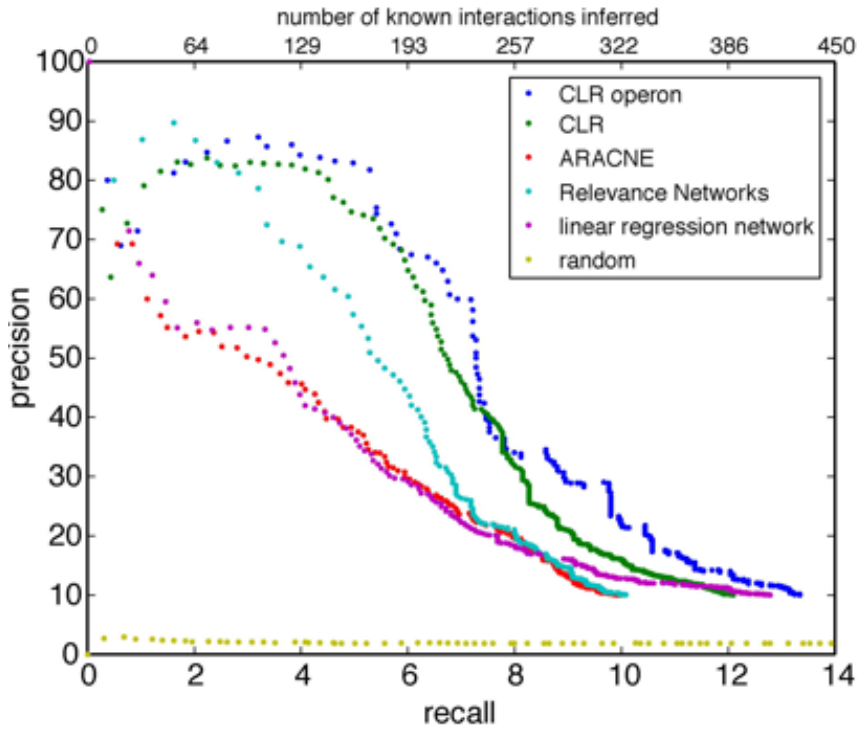
- For $t = 1$ to T do
  - Bootstrap a random sample S_t from the training set
  - Randomly reweight each feature
  - Select M features, e.g., with the Lassp
- The score of a feature is the number of times it was selected among the T repeats
- Rank features by decreasing score.
- See Meinshausen and Bühlmann (2009).

Examples of *de novo* methods

OPEN ACCESS Freely available online PLOS BIOLOGY

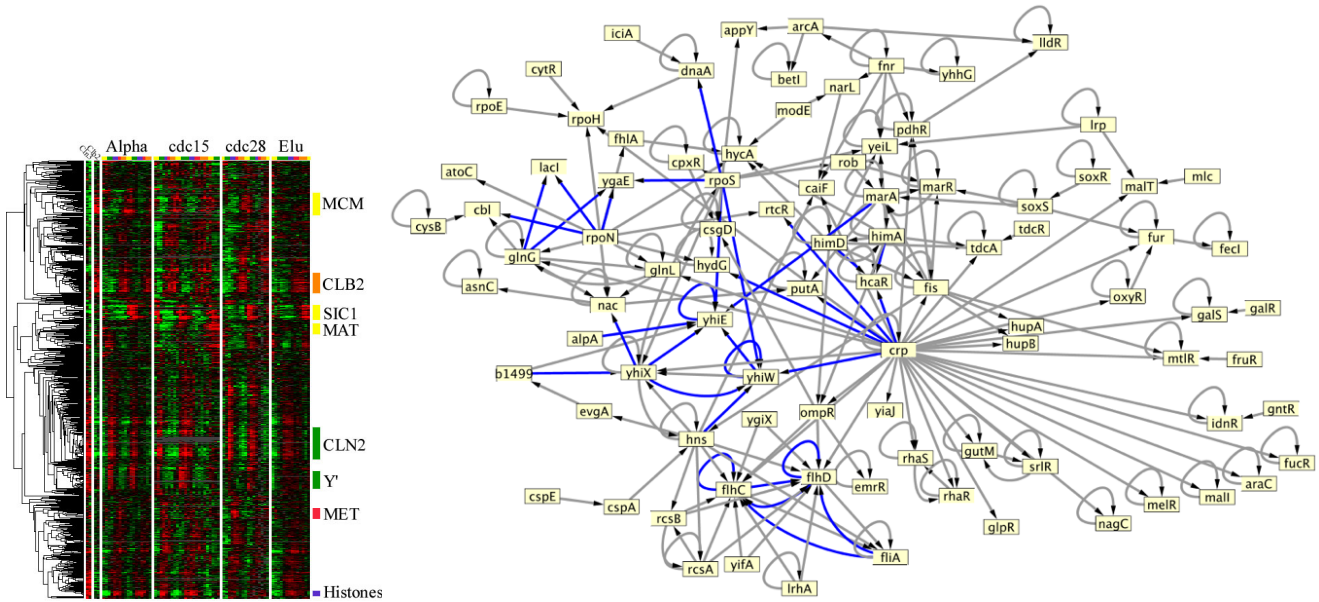
Large-Scale Mapping and Validation of *Escherichia coli* Transcriptional Regulation from a Compendium of Expression Profiles

Jeremiah J. Faith[1☯], Boris Hayete[1☯], Joshua T. Thaden[2,3], Ilaria Mogno[2,4], Jamey Wierzbowski[2,5], Guillaume Cottarel[2,5], Simon Kasif[1,2], James J. Collins[1,2], Timothy S. Gardner[1,2*]



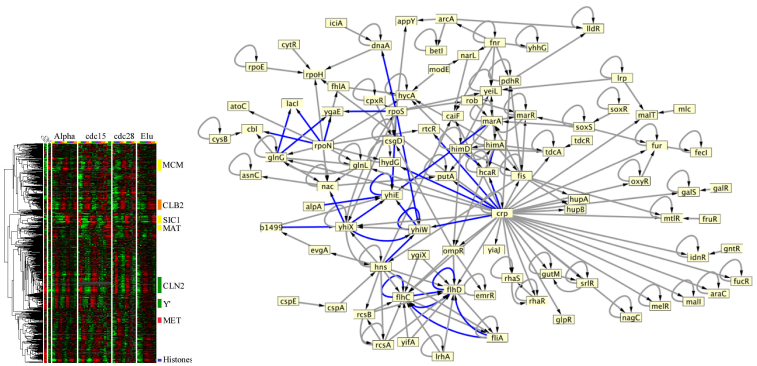
Outline

Motivations



- In many cases, we already know quite a few regulations.
- Can we use them, in addition to expression data, to *predict unknown regulations*?

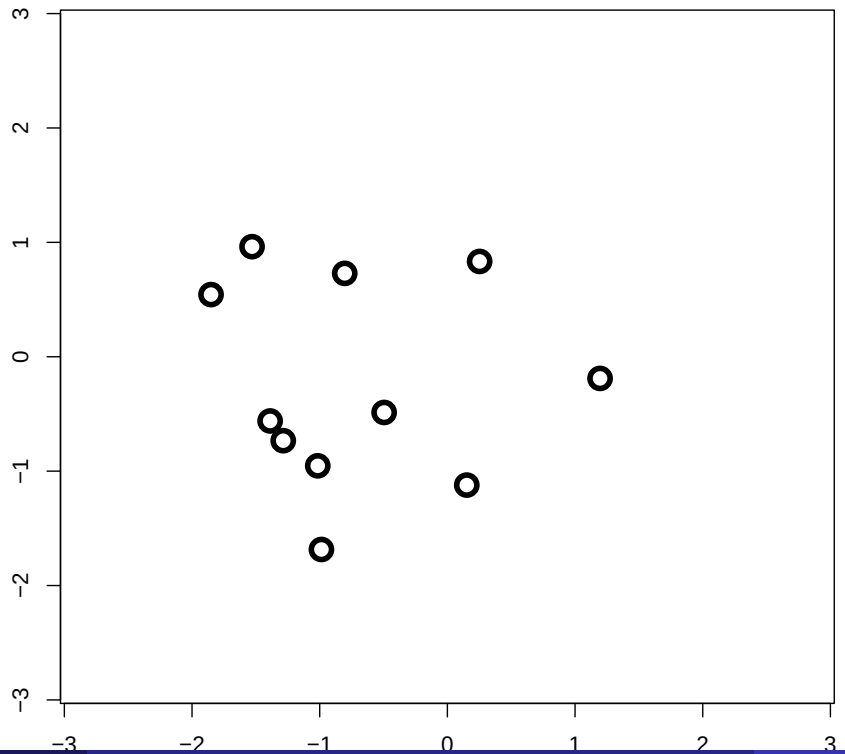
Using expression data for supervised inference



- If a gene has an expression profile similar to other genes known to be regulated by a TF, then it is likely to be regulated by the TF itself
- Underlying hypothesis: genes regulated by the same TF have similar expression variations
- Note that this is very different from *de novo* inference, where we compare the expression profile of the gene to that of the TF
- This is only possible if we already have a list of known regulations.

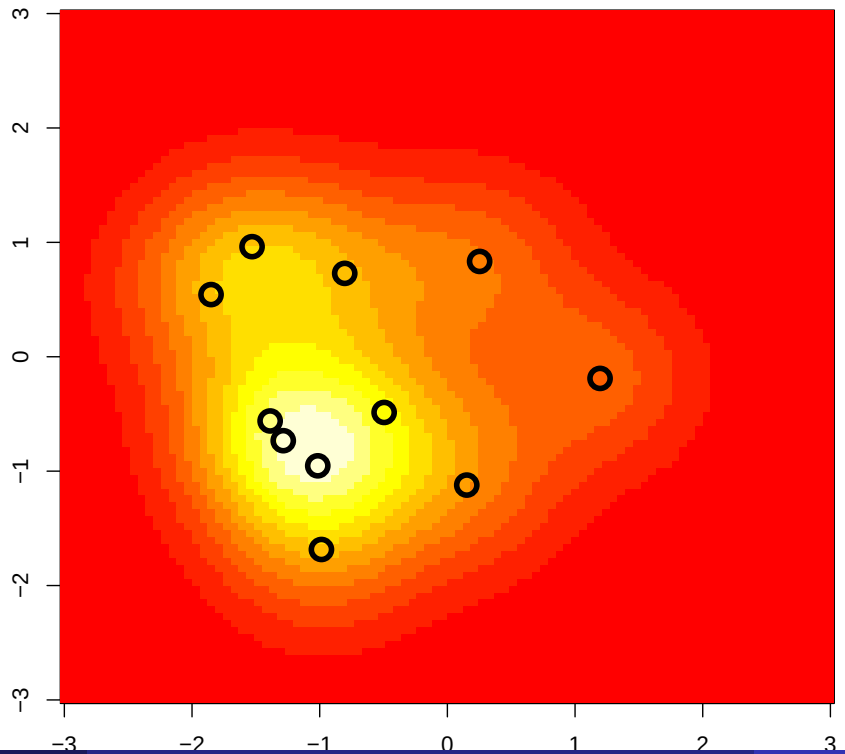
The idea

- For a given TF, let $P \subset [1, n]$ be the set of genes known to be regulated by it
- From the expression profiles $(X_i)_{i \in P}$, estimate a score $s(X)$ to assess which expression profiles X are similar
- Then classify the genes not in P by decreasing score



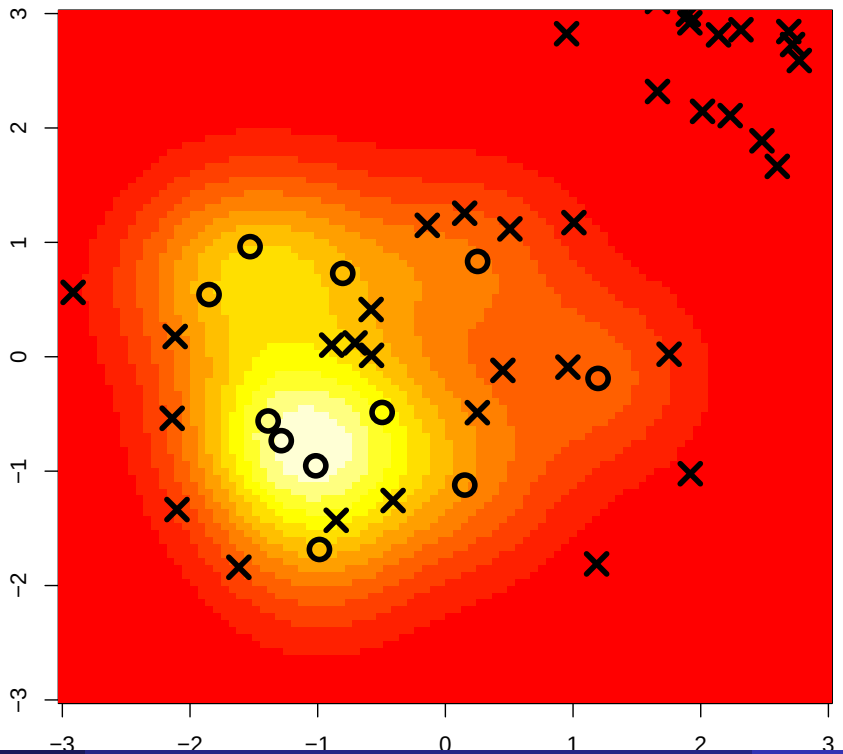
The idea

- For a given TF, let $P \subset [1, n]$ be the set of genes known to be regulated by it
- From the expression profiles $(X_i)_{i \in P}$, estimate a score $s(X)$ to assess which expression profiles X are similar
- Then classify the genes not in P by decreasing score

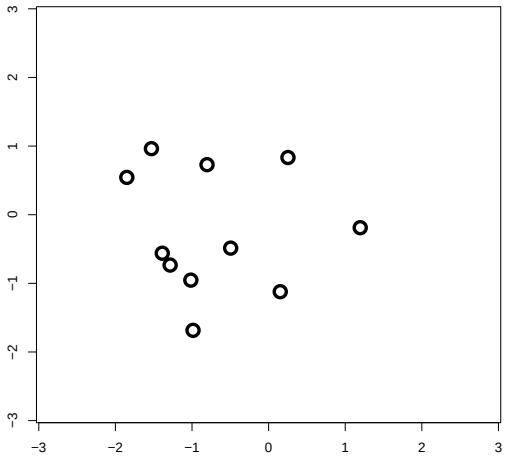


The idea

- For a given TF, let $P \subset [1, n]$ be the set of genes known to be regulated by it
- From the expression profiles $(X_i)_{i \in P}$, estimate a score $s(X)$ to assess which expression profiles X are similar
- Then classify the genes not in P by decreasing score



Estimating the scoring function: examples



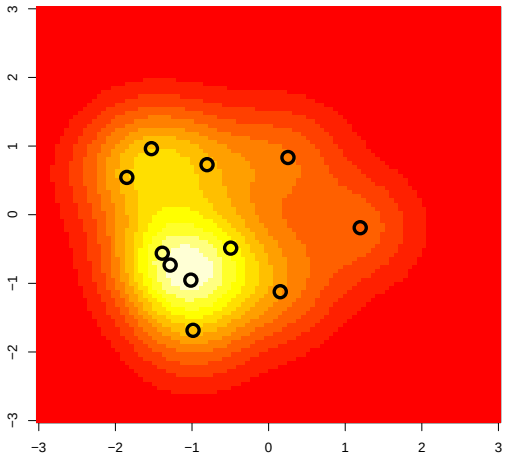
- Kernel density estimation

$$s(X) = \sum_{i \in P} \exp\left(-\gamma \| X - X_i \|^2\right)$$

- One-class SVM

$$s(X) = \sum_{i \in P} \alpha_i \exp\left(-\gamma \| X - X_i \|^2\right)$$

Estimating the scoring function: examples



- Kernel density estimation

$$s(X) = \sum_{i \in P} \exp\left(-\gamma \| X - X_i \|^2\right)$$

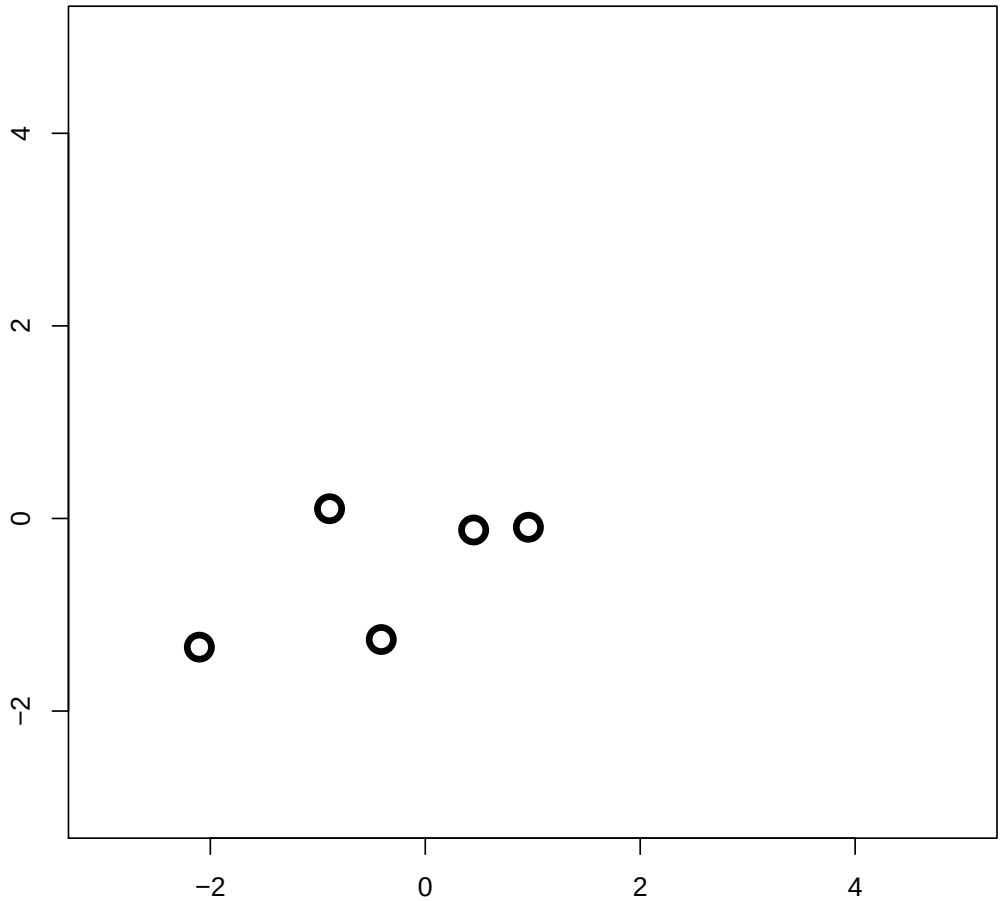
- One-class SVM

$$s(X) = \sum_{i \in P} \alpha_i \exp\left(-\gamma \| X - X_i \|^2\right)$$

Outline

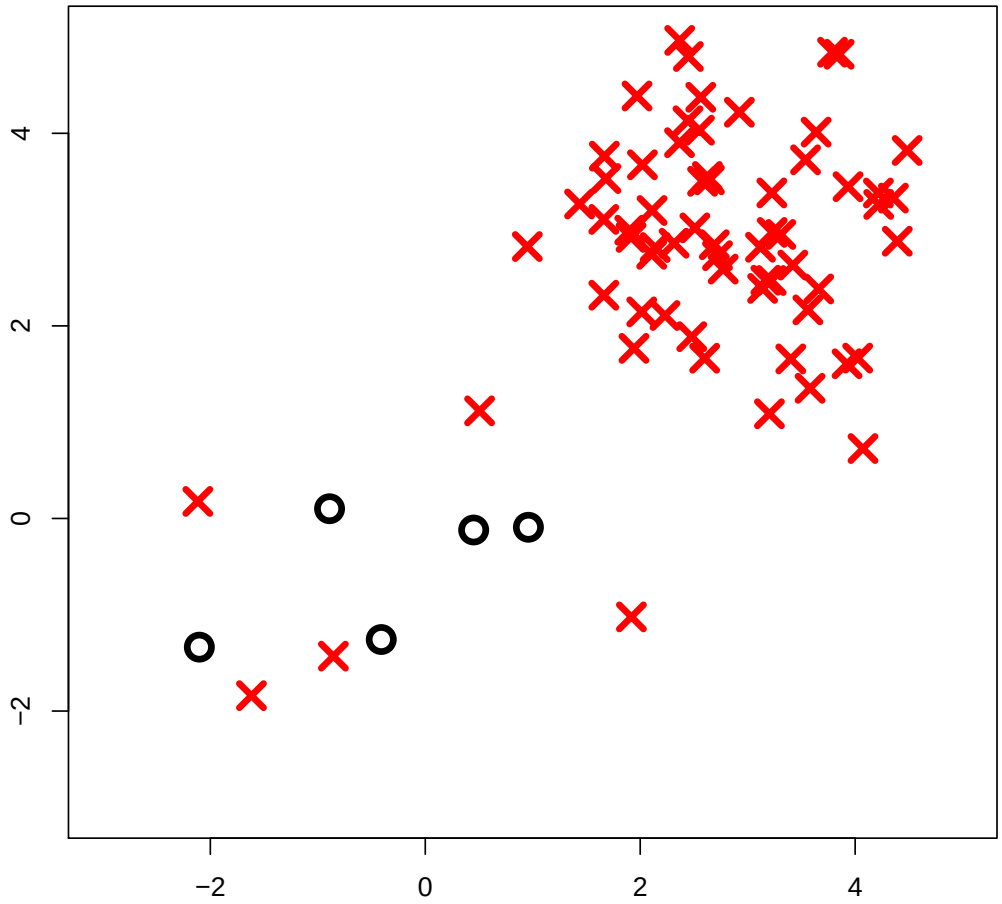
The idea

Since we know in advance all genes, can we use them instead of relying only on genes in P to estimate the scoring function?

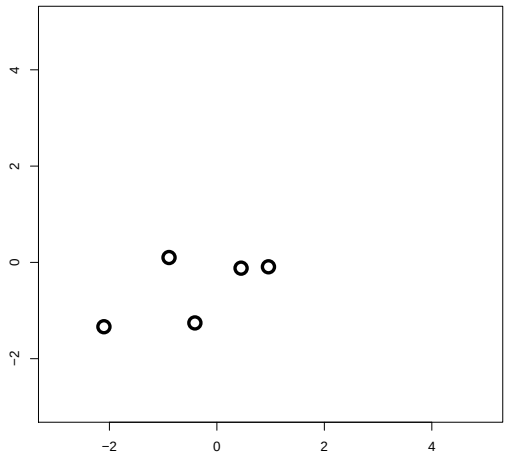


The idea

Since we know in advance all genes, can we use them instead of relying only on genes in P to estimate the scoring function?

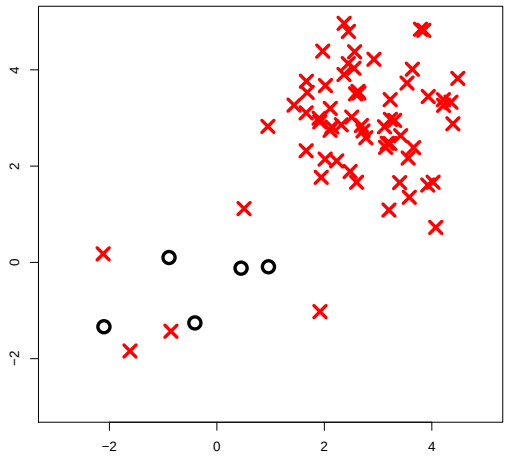


From one-class to PU learning



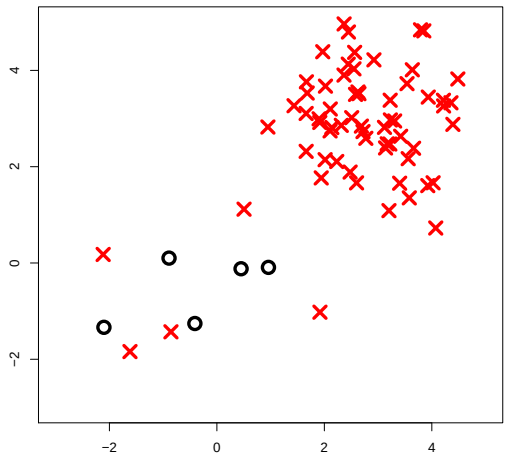
- One class: given genes in P, estimate the function $s(X)$
- PU learning: given genes in P and the set of unlabeled genes U, estimate the scores $s(X_j)$ for $j \in U$

From one-class to PU learning



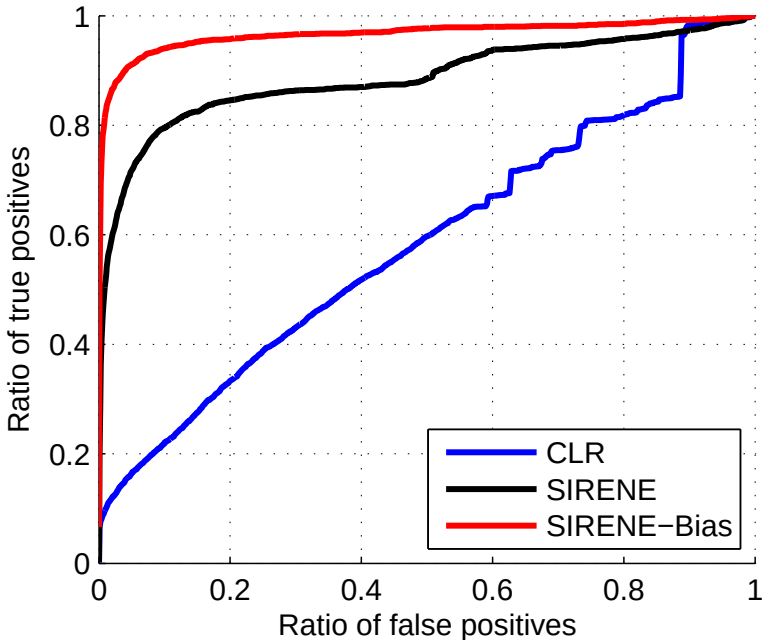
- One class: given genes in P, estimate the function $s(X)$
- PU learning: given genes in P and the set of unlabeled genes U, estimate the scores $s(X_j)$ for $j \in U$

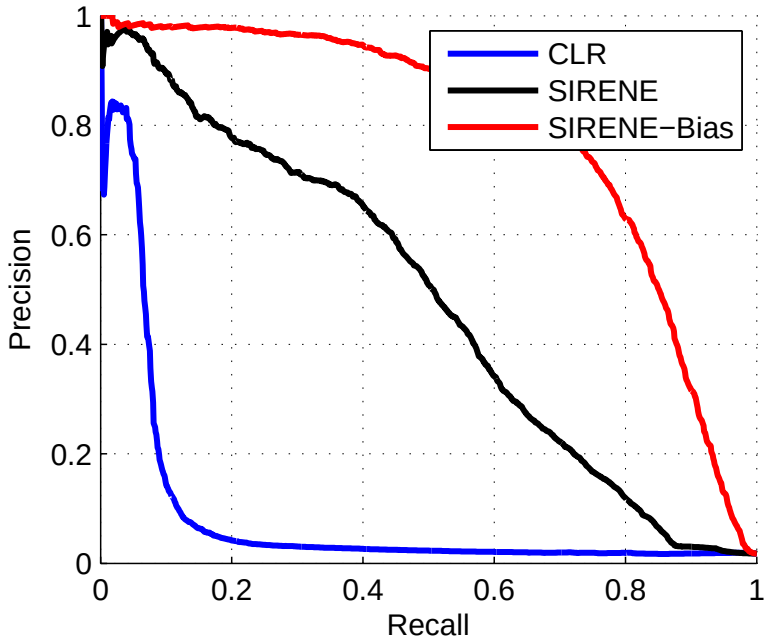
PU learning in practice



1. Train a classifier to discriminate P from U (eg, SVM or random forest)
2. Rank genes in U by decreasing training score

Example: E. coli regulatory network

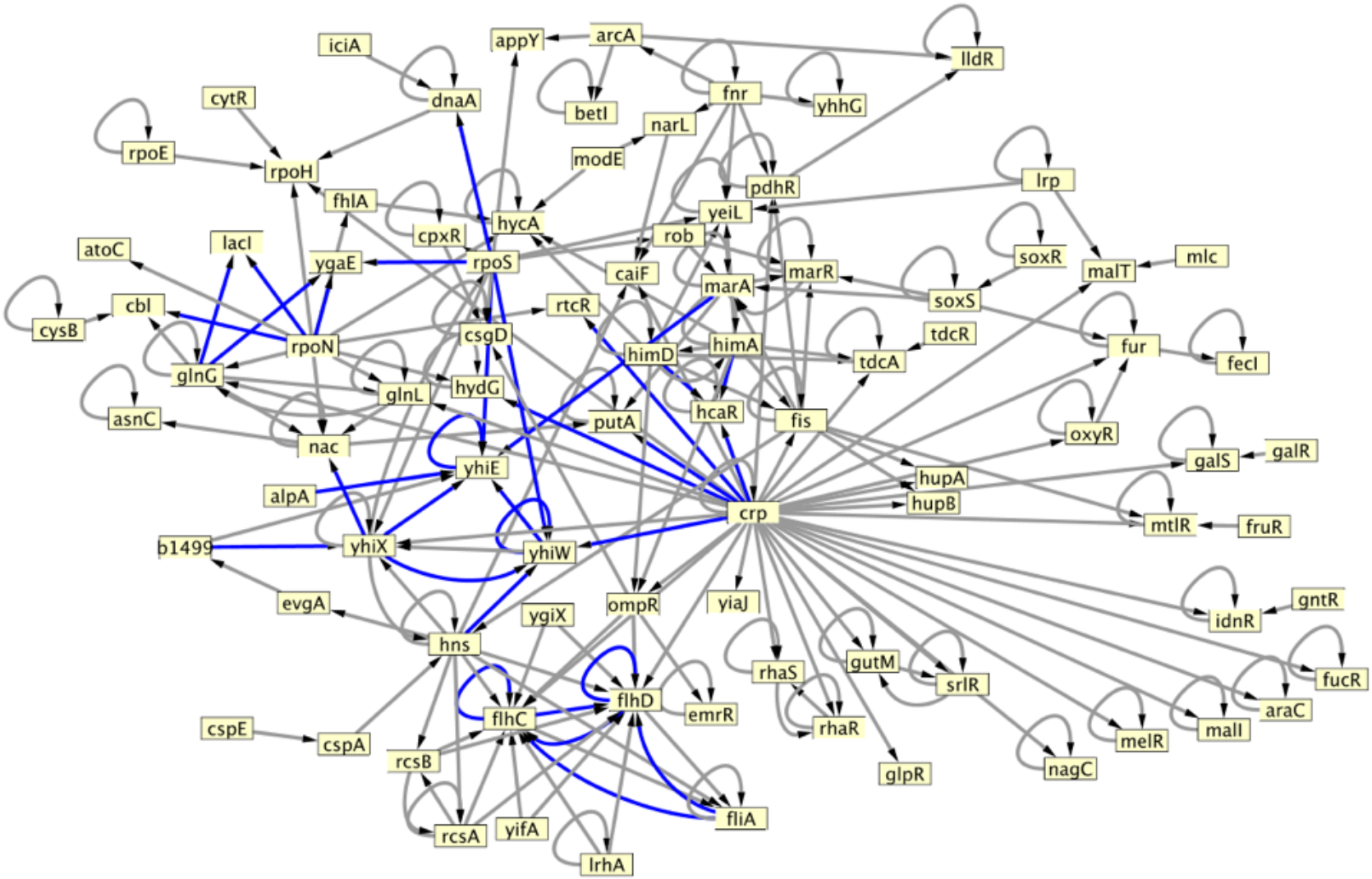




Method	Recall at 60%	Recall at 80%
SIRENE	**44.5%**	**17.6%**
CLR	7.5%	5.5%
Relevance networks	4.7%	3.3%
ARACNe	1%	0%
Bayesian network	1%	0%

SIRENE = Supervised Inference of REgulatory NEtworks (Mordelet and V., 2008)

Application: predicted regulatory network (E. coli)

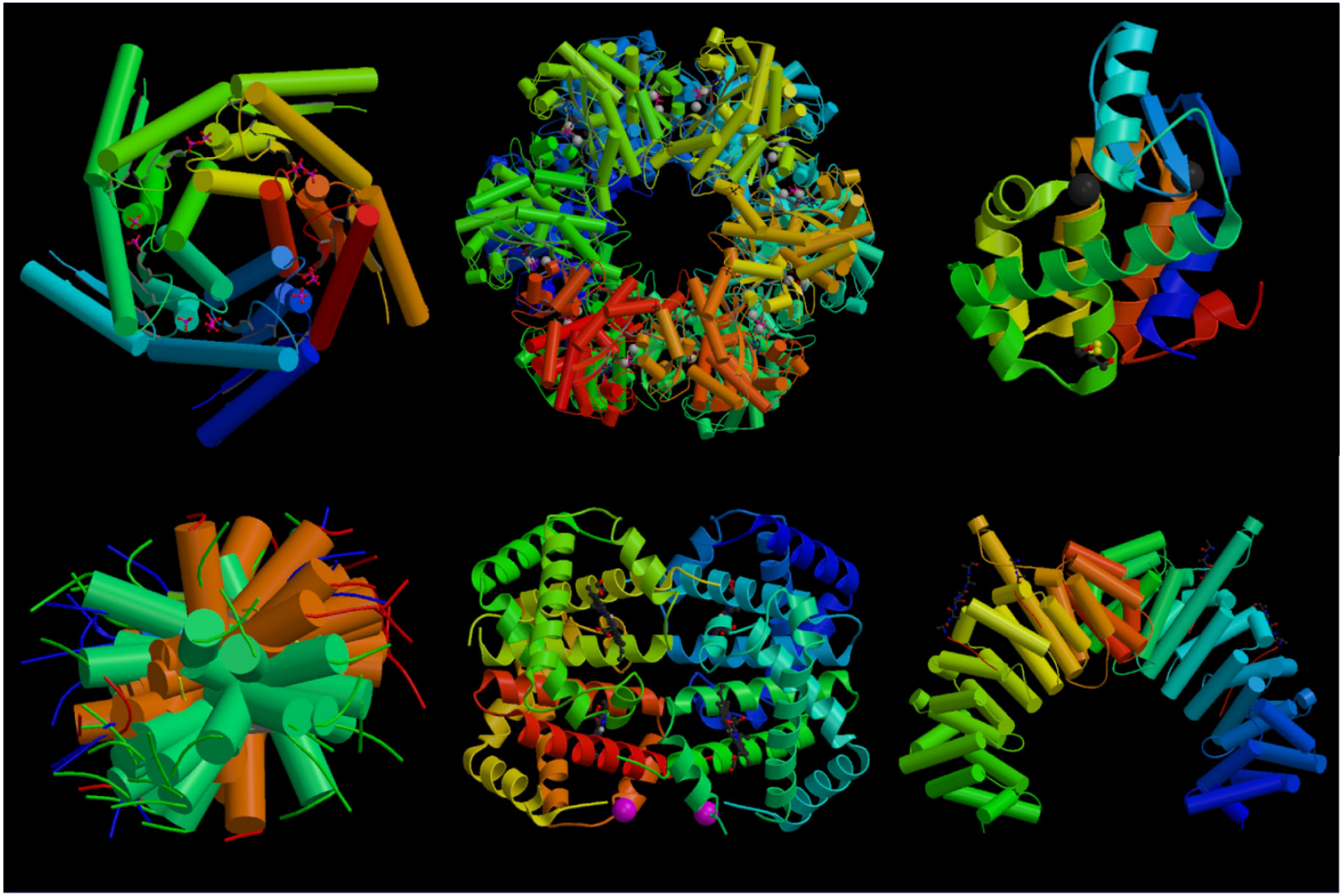


Prediction at 60% precision, restricted to transcription factors (from Mordelet and V., 2008).

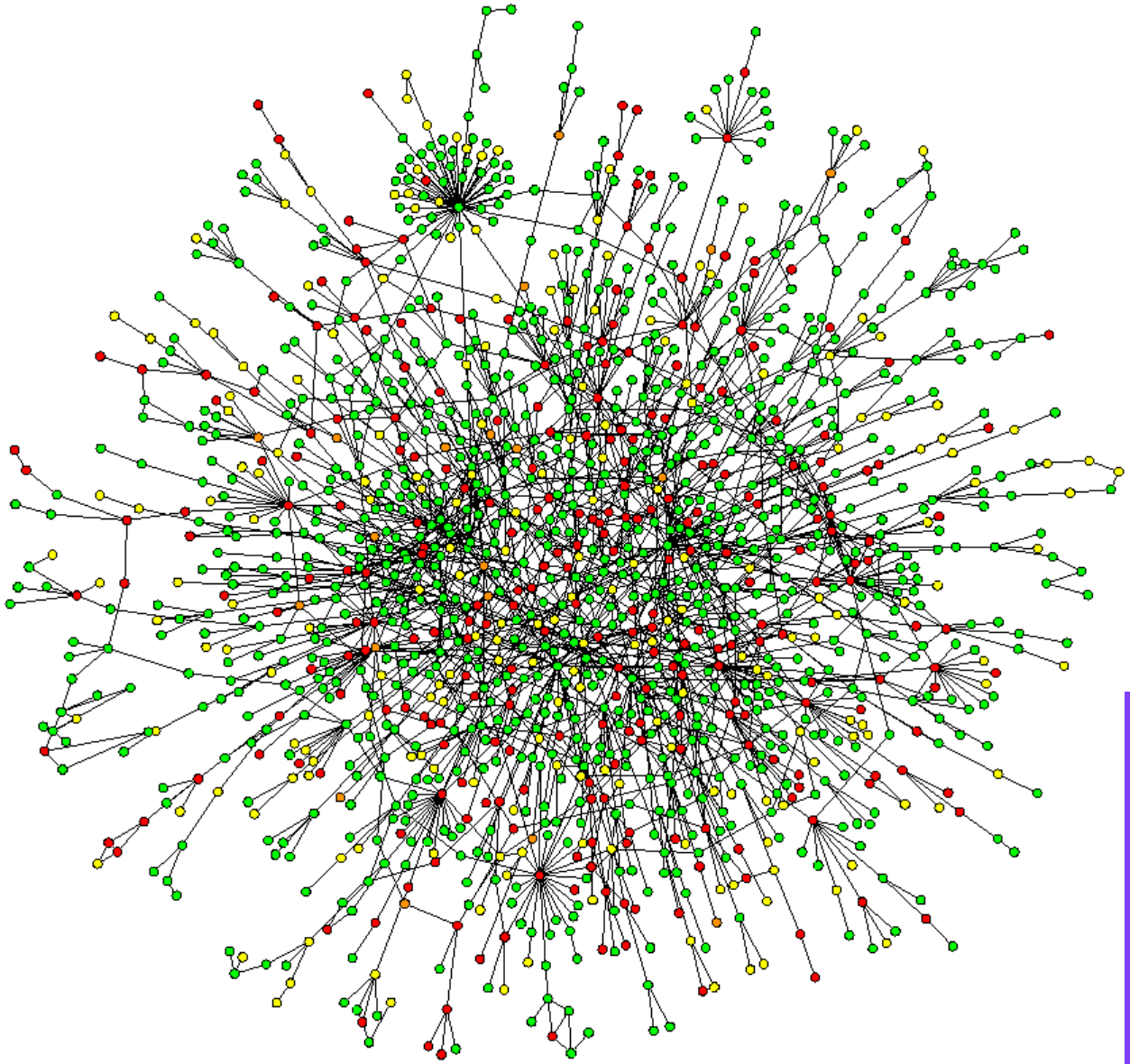
Outline

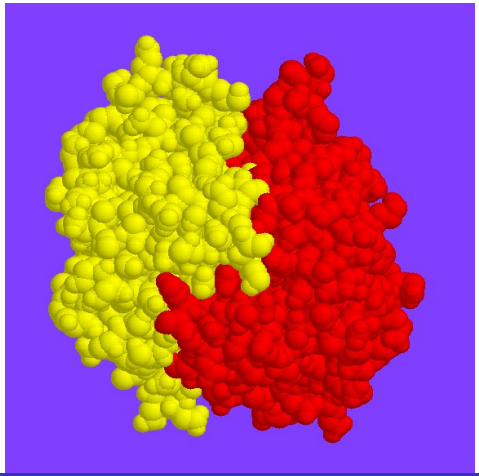
Outline

Proteins

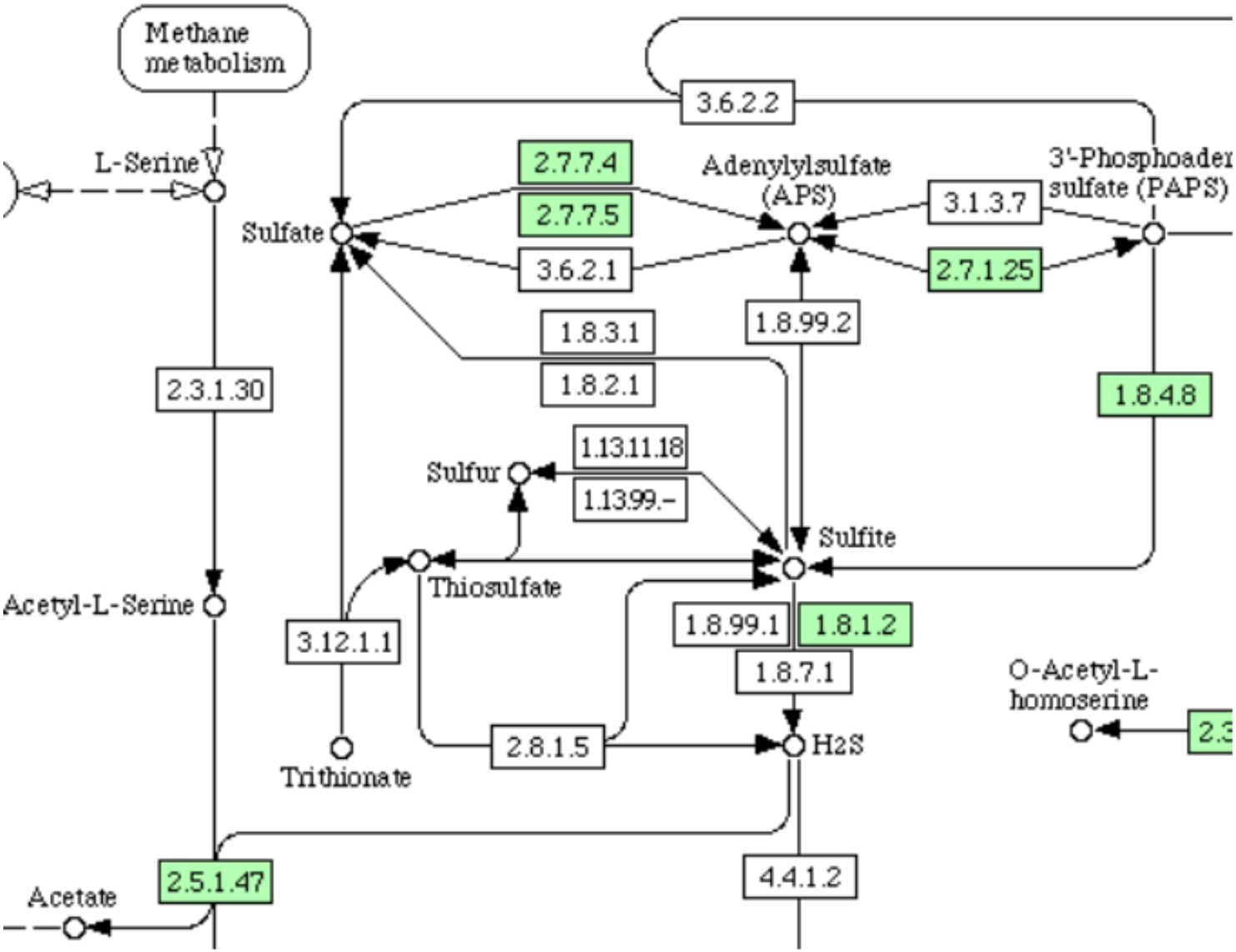


Network 1: protein-protein interaction

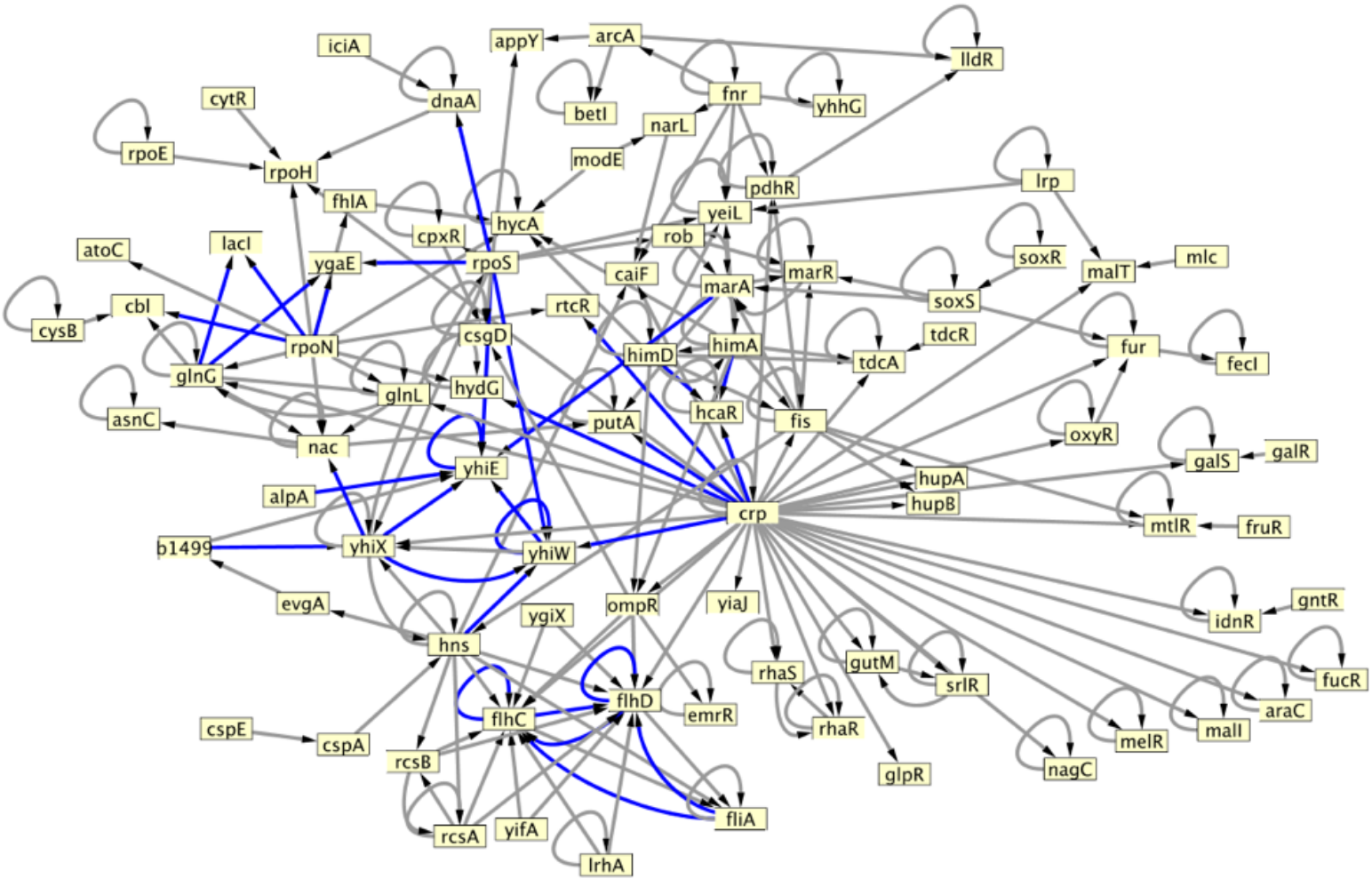




Network 2: metabolic network



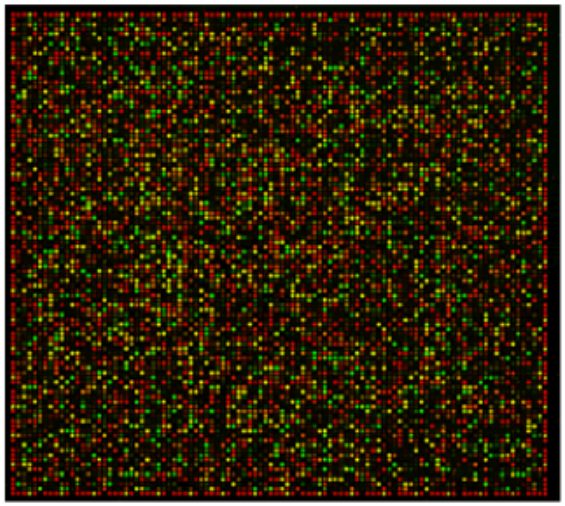
Network 3: gene regulatory network



Data available

Biologists have collected a lot of data about proteins. e.g.,

- Gene expression measurements
- Phylogenetic profiles
- Location of proteins/enzymes in the cell



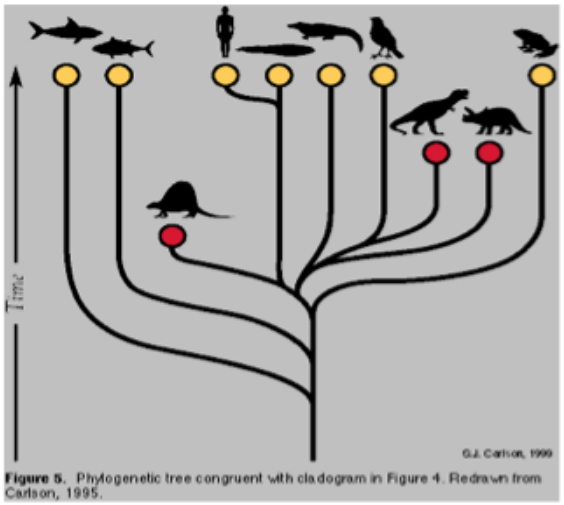
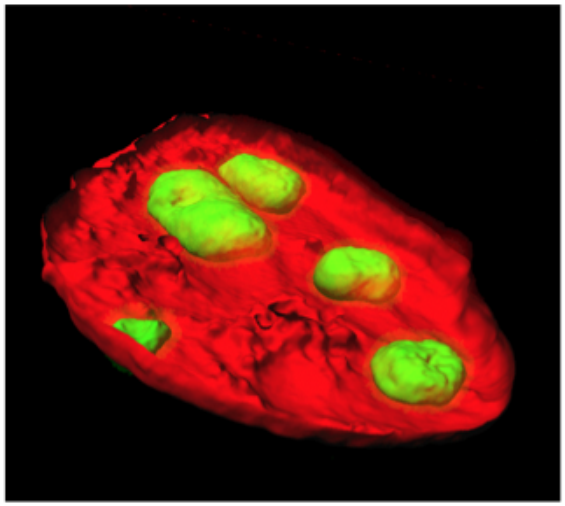
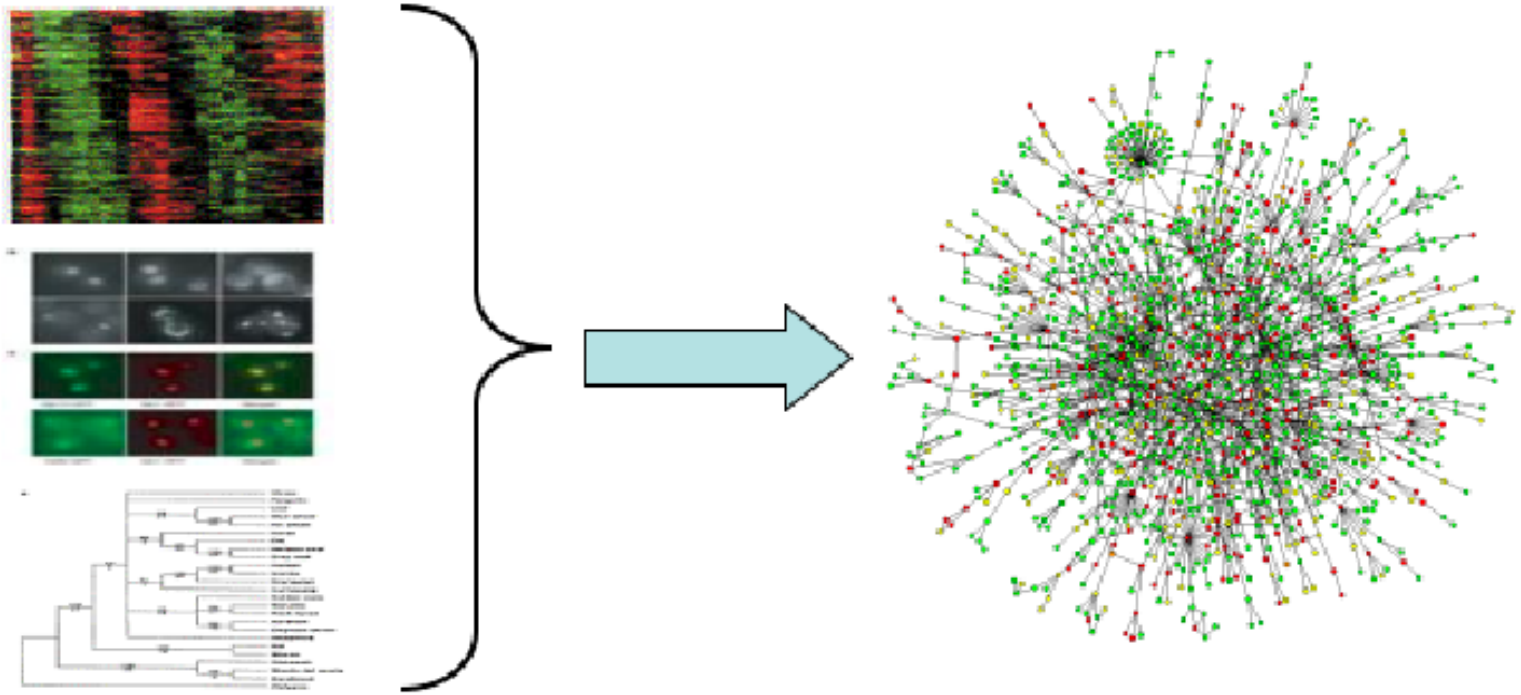


Figure 5. Phylogenetic tree congruent with cladogram in Figure 4. Redrawn from Carlson, 1995.



How to use this information “intelligently” to find a good function that predicts edges between nodes.

Our goal



Data

- Gene expression,
- Gene sequence,
- Protein localization, ...

Graph

- Protein-protein interactions,
- Metabolic pathways,
- Signaling pathways, ...

More precisely

Formalization

- $\mathcal{V} = \{1, \ldots, N\}$ vertices (*e.g., genes, proteins*)
- $\mathcal{D} = (x_1, \ldots, x_N) \in \mathcal{H}^N$ data about the vertices ($\mathcal{H}$ *Hilbert space*)
- Goal: predict edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$. We focus on undirected graphs.

"De novo" inference

- Given data about individual genes and proteins $\mathcal{D}$, ...
- ... Infer the edges between genes and proteins $\mathcal{E}$

"Supervised" inference

- Given data about individual genes and proteins $\mathcal{D}$, ...
- ... and given some known interactions $\mathcal{E}_{train} \subset \mathcal{E}$, ...
- ... infer unknown interactions $\mathcal{E}_{test} = \mathcal{E} \backslash \mathcal{E}_{train}$

More precisely

Formalization

- $\mathcal{V} = \{1, \ldots, N\}$ vertices (*e.g., genes, proteins*)
- $\mathcal{D} = (x_1, \ldots, x_N) \in \mathcal{H}^N$ data about the vertices ($\mathcal{H}$ *Hilbert space*)
- Goal: predict edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$. We focus on undirected graphs.

"De novo" inference

- Given data about individual genes and proteins $\mathcal{D}$, ...
- ... Infer the edges between genes and proteins $\mathcal{E}$

"Supervised" inference

- Given data about individual genes and proteins $\mathcal{D}$, ...
- ... and given some known interactions $\mathcal{E}_{train} \subset \mathcal{E}$, ...
- ... infer unknown interactions $\mathcal{E}_{test} = \mathcal{E} \backslash \mathcal{E}_{train}$

More precisely

Formalization

- $\mathcal{V} = \{1, \ldots, N\}$ vertices (*e.g., genes, proteins*)
- $\mathcal{D} = (x_1, \ldots, x_N) \in \mathcal{H}^N$ data about the vertices ($\mathcal{H}$ *Hilbert space*)
- Goal: predict edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$. We focus on undirected graphs.

"De novo" inference

- Given data about individual genes and proteins $\mathcal{D}$, ...
- ... Infer the edges between genes and proteins $\mathcal{E}$

"Supervised" inference

- Given data about individual genes and proteins $\mathcal{D}$, ...
- ... and given some known interactions $\mathcal{E}_{train} \subset \mathcal{E}$, ...
- ... infer unknown interactions $\mathcal{E}_{test} = \mathcal{E} \backslash \mathcal{E}_{train}$

De novo methods

Typical strategies

- Fit a dynamical system to time series (e.g., PDE, boolean networks, state-space models)
- Detect statistical conditional independence or dependency (Bayesian netwok, mutual information networks, co-expression)

Pros

- Excellent approach if the model is correct and enough data are available
- Interpretability of the model
- Inclusion of prior knowledge

Cons

- Specific to particular data and networks
- Needs a correct model!
- Difficult integration of heterogeneous data
- Often needs a lot of data and long computation time

De novo methods

Typical strategies

- Fit a dynamical system to time series (e.g., PDE, boolean networks, state-space models)
- Detect statistical conditional independence or dependency (Bayesian netwok, mutual information networks, co-expression)

Pros

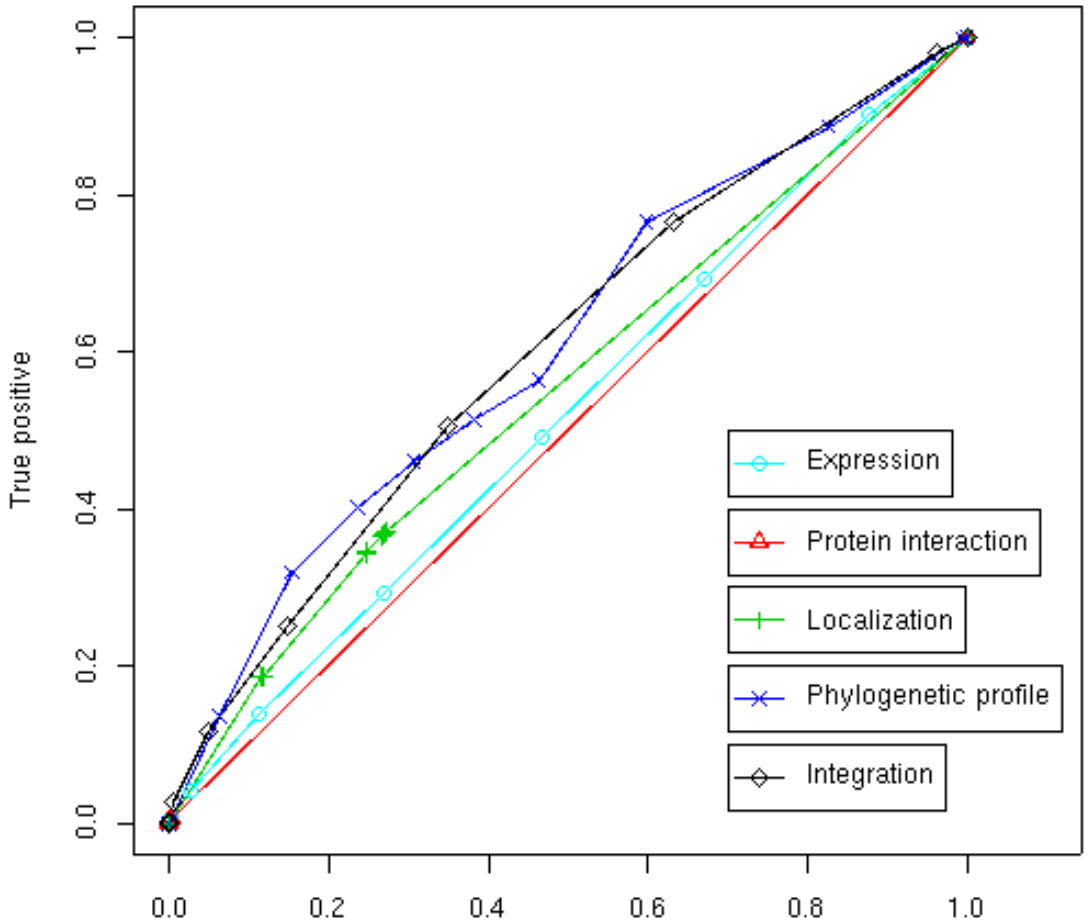
- Excellent approach if the model is correct and enough data are available
- Interpretability of the model
- Inclusion of prior knowledge

Cons

- Specific to particular data and networks
- Needs a correct model!
- Difficult integration of heterogeneous data
- Often needs a lot of data and long computation time

Evaluation on metabolic network reconstruction

- The known metabolic network of the yeast involves 769 proteins.
- Predict edges from distances between a variety of genomic data (expression, localization, phylogenetic profiles, interactions).

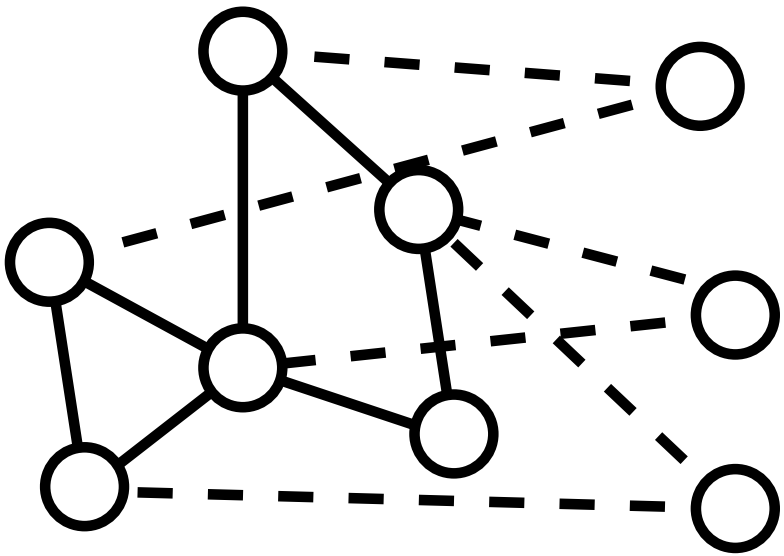


Supervised methods

Motivation

In actual applications,

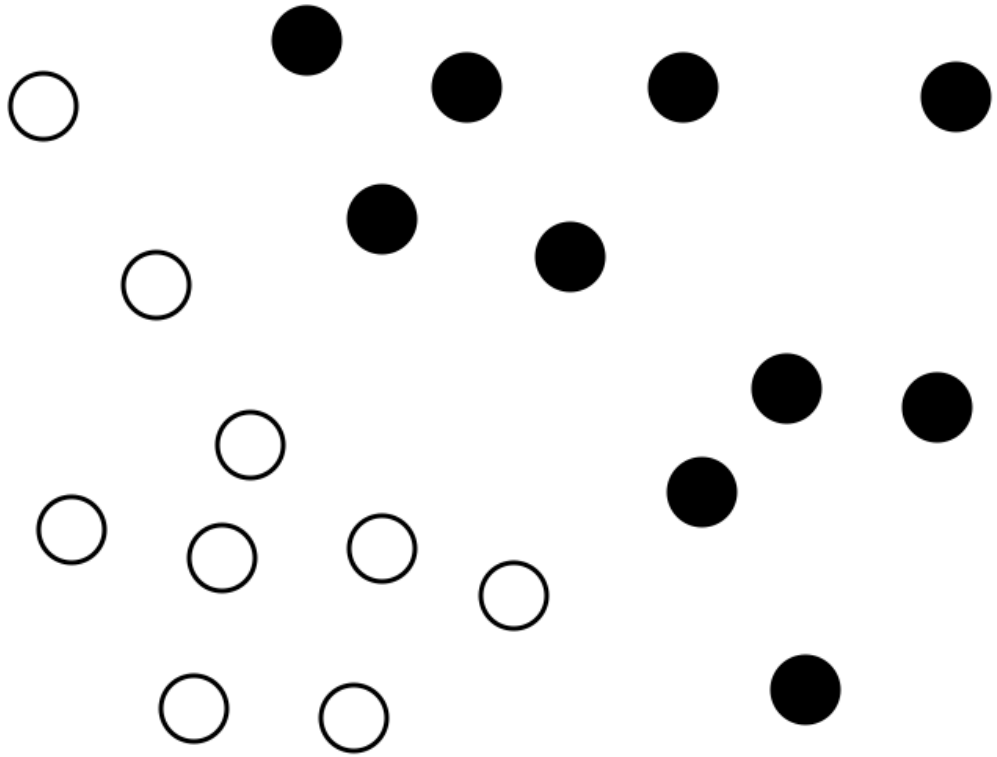
- we know in advance parts of the network to be inferred
- the problem is to add/remove nodes and edges using genomic data as side information



Supervised method

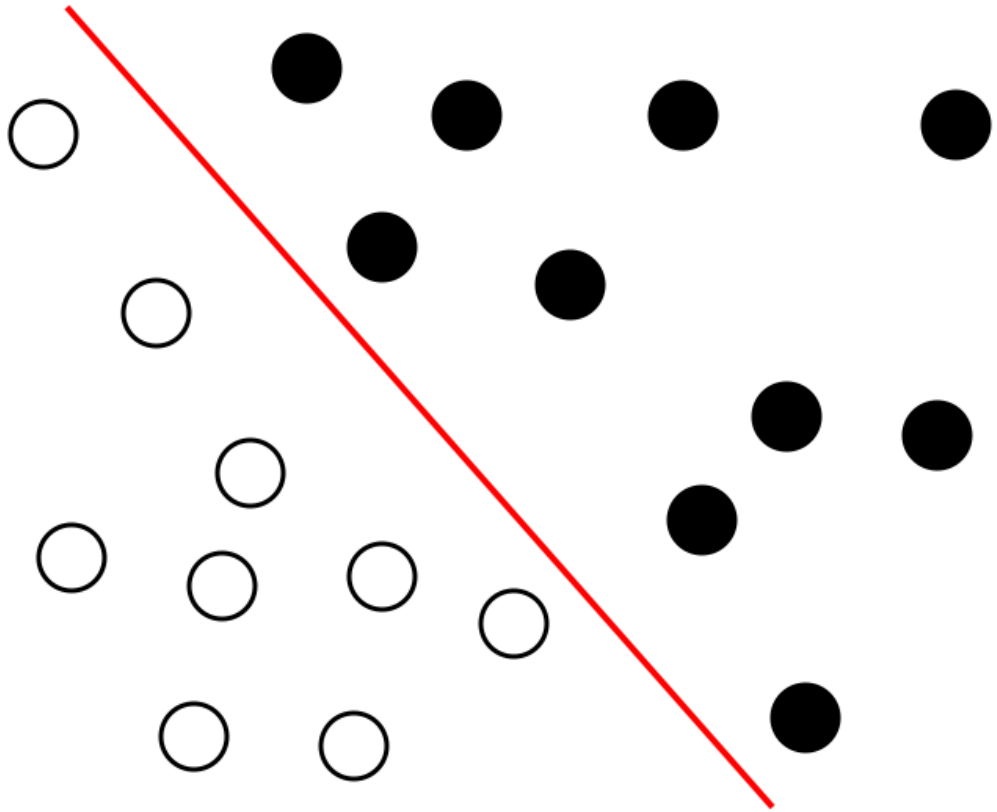
- Given genomic data and the currently known network...
- Infer missing edges between current nodes and additional nodes.

Pattern recognition



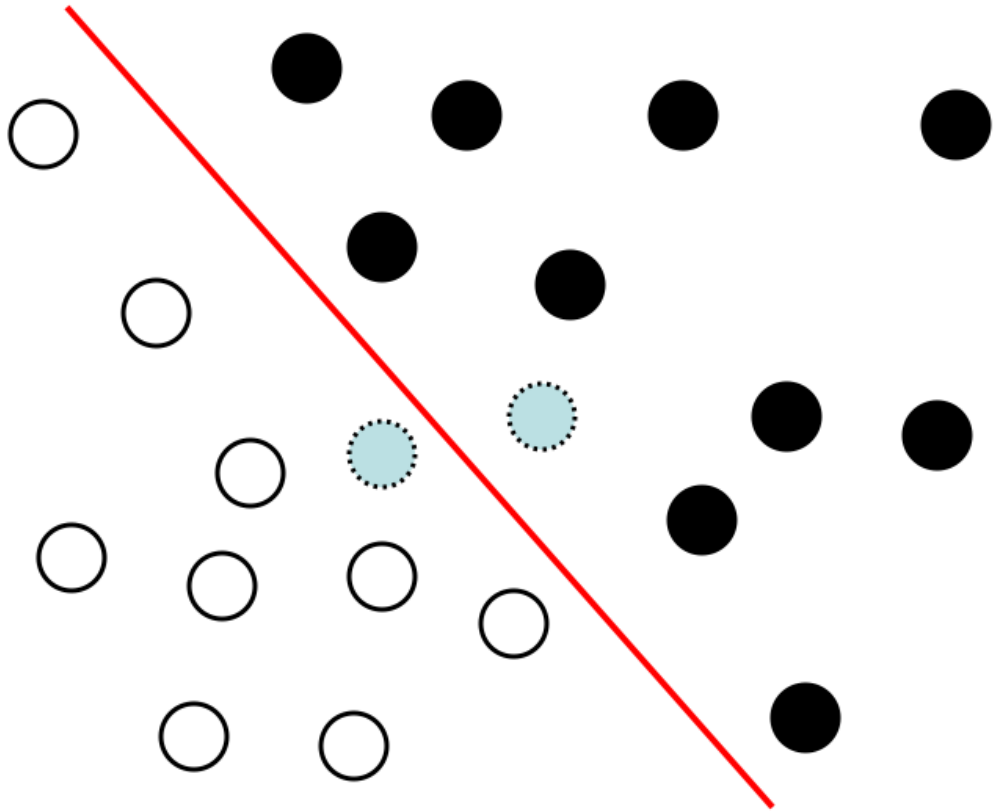
- Given a training set of patterns in two classes, learn to discriminate them
- Many algorithms (ANN, SVM, Decision tress, ...)

Pattern recognition



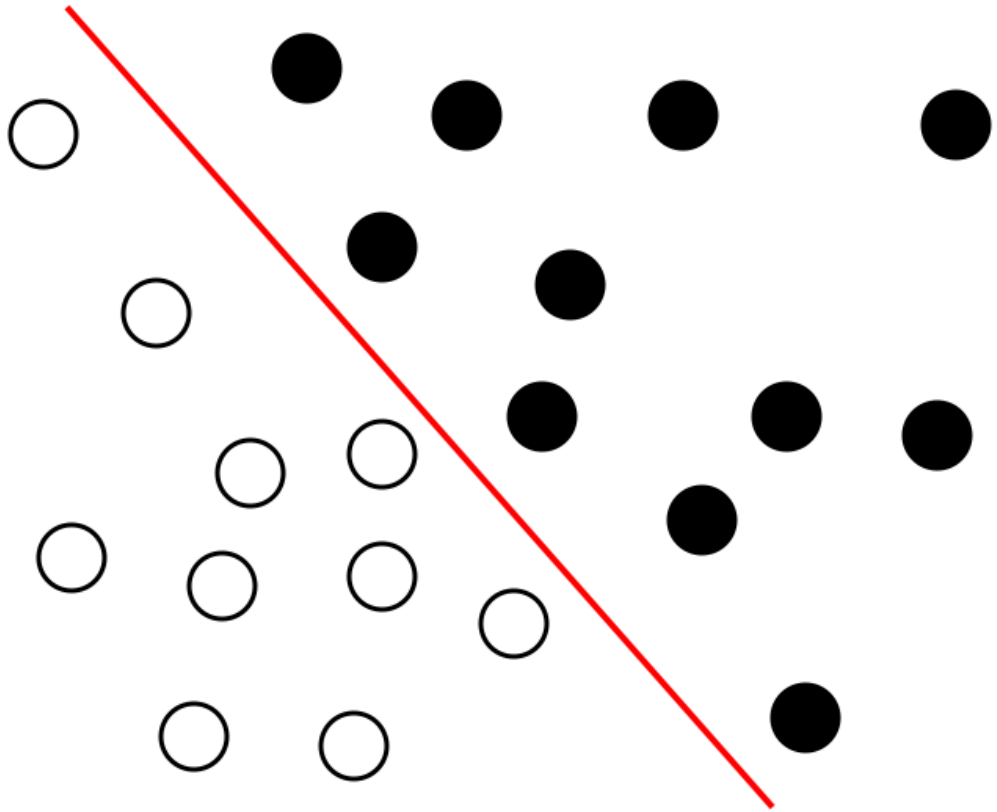
- Given a training set of patterns in two classes, learn to discriminate them
- Many algorithms (ANN, SVM, Decision tress, ...)

Pattern recognition



- Given a training set of patterns in two classes, learn to discriminate them
- Many algorithms (ANN, SVM, Decision tress, ...)

Pattern recognition



- Given a training set of patterns in two classes, learn to discriminate them
- Many algorithms (ANN, SVM, Decision tress, ...)

Pattern recognition and graph inference

Pattern recognition

Associate a binary label Y to each data X

Graph inference

Associate a binary label Y to each pair of data (X_1, X_2)

Two solutions

- Consider each pair (X_1, X_2) as a single data -> learning over pairs
- Reformulate the graph inference problem as a pattern recognition problem at the level of individual vertices -> local models

Pattern recognition and graph inference

Pattern recognition

Associate a binary label Y to each data X

Graph inference

Associate a binary label Y to each pair of data (X_1, X_2)

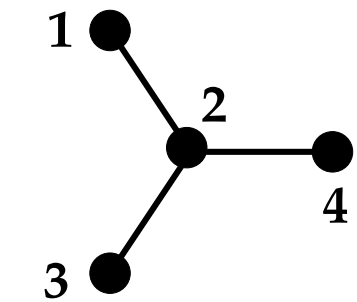
Two solutions

- Consider each pair (X_1, X_2) as a single data -> learning over pairs
- Reformulate the graph inference problem as a pattern recognition problem at the level of individual vertices -> local models

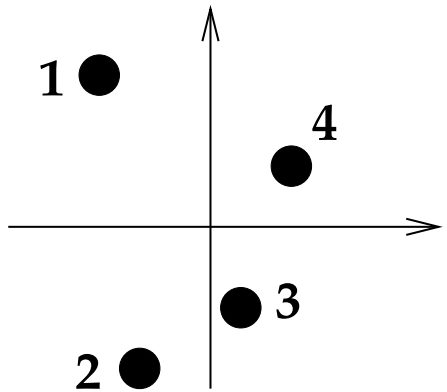
Outline

Pattern recognition for pairs: basic issue

- A pair can be connected (1) or not connected (-1)
- From the known subgraph we can extract examples of connected and non-connected pairs
- However the genomic data characterize individual proteins; we need to work with pairs of proteins instead!



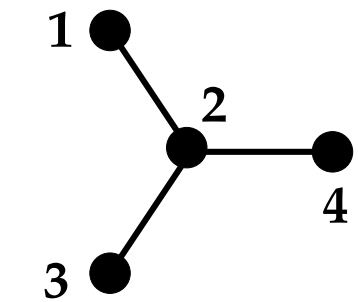
Known graph



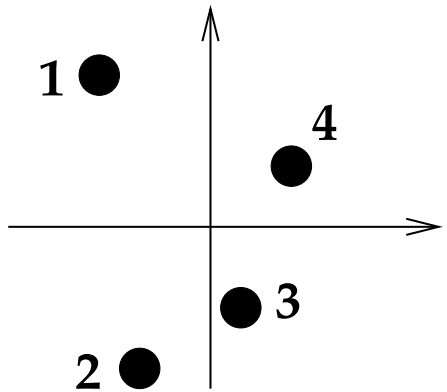
Genomic data

Pattern recognition for pairs: basic issue

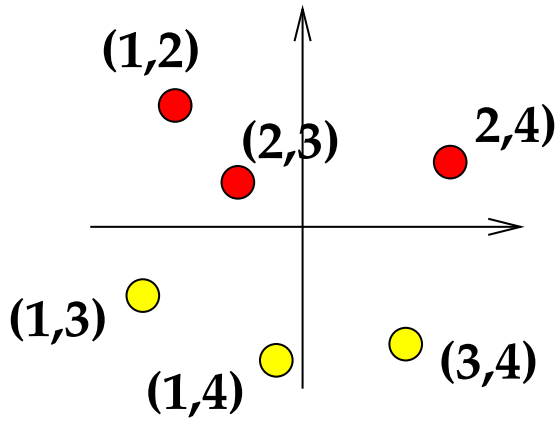
- A pair can be connected (1) or not connected (-1)
- From the known subgraph we can extract examples of connected and non-connected pairs
- However the genomic data characterize individual proteins; we need to work with pairs of proteins instead!



Known graph

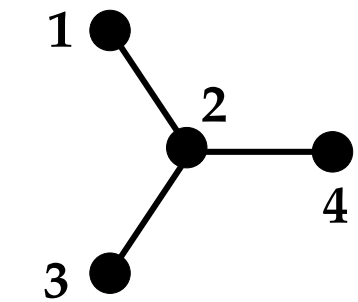


Genomic data

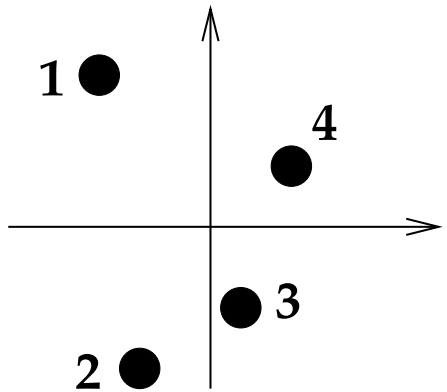


Pattern recognition for pairs: basic issue

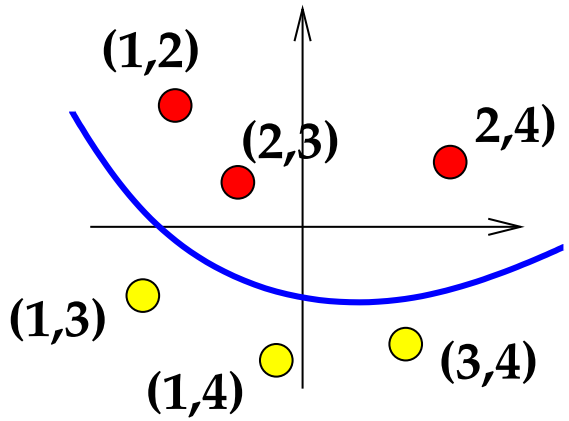
- A pair can be connected (1) or not connected (-1)
- From the known subgraph we can extract examples of connected and non-connected pairs
- However the genomic data characterize individual proteins; we need to work with pairs of proteins instead!



Known graph



Genomic data



Representing a pair as a vector

- Each individual protein is represented by a vector $v \in \mathbb{R}^p$
- Depending on the network, we are interested in ordered or unordered pairs of proteins.
- We must represent a pair of proteins (u, v) by a vector $\psi(u, v) \in \mathbb{R}^q$ in order to estimate a linear classifier
- Question: how build $\psi(u, v)$ from u and v, in the ordered and unordered cases?

Direct sum for ordered pairs?

- A simple idea is to concatenate the vectors u and v to obtain a $2p$-dimensional vector of (u, v):

$$\psi(u, v) = u \oplus v = \begin{pmatrix} u \\ v \end{pmatrix} .$$

- Problem: a linear function then becomes additive...

$$f(u, v) = w^\top \psi(u, v) = w_1^\top u + w^\top v .$$

Direct sum for ordered pairs?

- A simple idea is to concatenate the vectors u and v to obtain a $2p$-dimensional vector of (u, v):

$$\psi(u, v) = u \oplus v = \begin{pmatrix} u \\ v \end{pmatrix} .$$

- Problem: a linear function then becomes additive...

$$f(u, v) = w^\top \psi(u, v) = w_1^\top u + w^\top v .$$

Direct product for ordered pairs

- Alternatively, make the direct product, i.e., the p^2-dimensional vector whose entries are all products of entries of u by entries of v:

$$\psi(u, v) = u \otimes v$$

- Problem: can get really large-dimensional...
- Good news: inner product factorizes:

$$(u_1 \otimes v_1)^\top (u_2 \otimes v_2) = \left(u_1^\top u_2\right) \times \left(v_1^\top v_2\right) ,$$

which is good for algorithms that use only inner products (SVM...):

$$K_P((u_1, v_1), (u_2, v_2)) = \psi(u_1, v_1)^\top \psi(u_2, v_2) = K(u_1, u_2) K(v_1, v_2)$$

Direct product for ordered pairs

- Alternatively, make the direct product, i.e., the p^2-dimensional vector whose entries are all products of entries of u by entries of v:

$$\psi(u, v) = u \otimes v$$

- Problem: can get really large-dimensional...
- Good news: inner product factorizes:

$$(u_1 \otimes v_1)^\top (u_2 \otimes v_2) = \left(u_1^\top u_2\right) \times \left(v_1^\top v_2\right) ,$$

which is good for algorithms that use only inner products (SVM...):

$$K_P((u_1, v_1), (u_2, v_2)) = \psi(u_1, v_1)^\top \psi(u_2, v_2) = K(u_1, u_2) K(v_1, v_2)$$

Direct product for ordered pairs

- Alternatively, make the direct product, i.e., the p^2-dimensional vector whose entries are all products of entries of u by entries of v:

$$\psi(u, v) = u \otimes v$$

- Problem: can get really large-dimensional...
- Good news: inner product factorizes:

$$(u_1 \otimes v_1)^\top (u_2 \otimes v_2) = \left(u_1^\top u_2\right) \times \left(v_1^\top v_2\right) ,$$

which is good for algorithms that use only inner products (SVM...):

$$K_P((u_1, v_1), (u_2, v_2)) = \psi(u_1, v_1)^\top \psi(u_2, v_2) = K(u_1, u_2) K(v_1, v_2)$$

Representing an unordered pair

- Often we want to work with unordered pairs, e.g., PPI network:

$$\{u, v\} = \{(u, v), (v, u)\}$$

- This suggest to symmetrize the representation of ordered pairs:

$$\psi_U(\{u, v\}) = \psi(u, v) + \psi(v, u)$$

- When $\psi(u, v) = u \otimes v$, this leads to the symmetric tensor product pairwise kernel (TPPK) (Ben-Hur and Noble, 2006):

$$K_{TPPK}\left(\{u_1, v_1\}, \{u_2, v_2\}\right) = K(u_1, u_2)K(v_1, v_2) + K(u_1, v_2)K(v_1, u_2)$$

Representing an unordered pair

- Often we want to work with unordered pairs, e.g., PPI network:

$$\{u, v\} = \{(u, v), (v, u)\}$$

- This suggest to symmetrize the representation of ordered pairs:

$$\psi_U(\{u, v\}) = \psi(u, v) + \psi(v, u)$$

- When $\psi(u, v) = u \otimes v$, this leads to the symmetric tensor product pairwise kernel (TPPK) (Ben-Hur and Noble, 2006):

$$K_{TPPK}\left(\{u_1, v_1\}, \{u_2, v_2\}\right) = K(u_1, u_2)K(v_1, v_2) + K(u_1, v_2)K(v_1, u_2)$$

Representing an unordered pair

- Often we want to work with unordered pairs, e.g., PPI network:

$$\{u, v\} = \{(u, v), (v, u)\}$$

- This suggest to symmetrize the representation of ordered pairs:

$$\psi_U(\{u, v\}) = \psi(u, v) + \psi(v, u)$$

- When $\psi(u, v) = u \otimes v$, this leads to the symmetric tensor product pairwise kernel (TPPK) (Ben-Hur and Noble, 2006):

$$K_{TPPK}\left(\{u_1, v_1\}, \{u_2, v_2\}\right) = K(u_1, u_2)K(v_1, v_2) + K(u_1, v_2)K(v_1, u_2)$$

Another idea: metric learning

- For two vectors $u, v \in \mathcal{H}$ let the metric:

$$d_M(u, v) = (u - v)^\top M(u - v)\,.$$

- Can we learn the metric M such that, in the new metric, connected points are near each other, and non-connected points are far from each other?
- We consider the problem:

$$\min_{M \geq 0} \sum_i l(u_i, v_i, y_i) + \lambda ||M||^2_{Frobenius}\,,$$

where l is a *hinge loss* to enforce:

$$d_M(u_i, v_i) \begin{cases} \leq 1 - \gamma & \text{if}(u_i, v_i)\text{is connected}\,, \\ \geq 1 + \gamma & \text{otherwise.} \end{cases}$$

Another idea: metric learning

- For two vectors $u, v \in \mathcal{H}$ let the metric:

$$d_M(u, v) = (u - v)^\top M(u - v)\,.$$

- Can we learn the metric M such that, in the new metric, connected points are near each other, and non-connected points are far from each other?
- We consider the problem:

$$\min_{M \geq 0} \sum_i l(u_i, v_i, y_i) + \lambda ||M||^2_{Frobenius}\,,$$

where l is a *hinge loss* to enforce:

$$d_M(u_i, v_i) \begin{cases} \leq 1 - \gamma & \text{if}(u_i, v_i)\text{is connected}\,, \\ \geq 1 + \gamma & \text{otherwise.} \end{cases}$$

Another idea: metric learning

- For two vectors $u, v \in \mathcal{H}$ let the metric:

$$d_M(u, v) = (u - v)^\top M(u - v)\,.$$

- Can we learn the metric M such that, in the new metric, connected points are near each other, and non-connected points are far from each other?
- We consider the problem:

$$\min_{M \geq 0} \sum_i l(u_i, v_i, y_i) + \lambda ||M||^2_{Frobenius}\,,$$

where l is a *hinge loss* to enforce:

$$d_M(u_i, v_i) \begin{cases} \leq 1 - \gamma & \text{if}(u_i, v_i)\text{is connected}\,, \\ \geq 1 + \gamma & \text{otherwise.} \end{cases}$$

Link with metric learning

Theorem (V. et al., 2007)

- A SVM with the representation

$$\psi(\{u, v\}) = (u - v)^{\otimes 2}$$

trained to discriminate connected from non-connected pairs, solves this metric learning problem without the constraint $M \geq 0$.

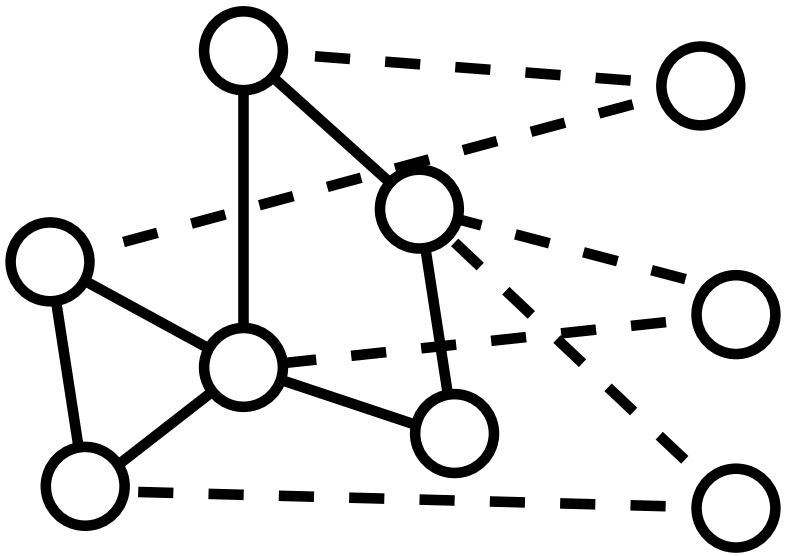
- Equivalently, train the SVM over pairs with the metric learning pairwise kernel:

$$\begin{aligned} K_{MLPK}\left(\{u_1, v_1\}, \{u_2, v_2\}\right) &= \psi(\{u_1, v_1\})^\top \psi(\{u_2, v_2\}) \\ &= \left[K(u_1, u_2) - K(u_1, v_2) - K(v_1, u_2) + K(u_2, v_2)\right]^2 . \end{aligned}$$

Outline

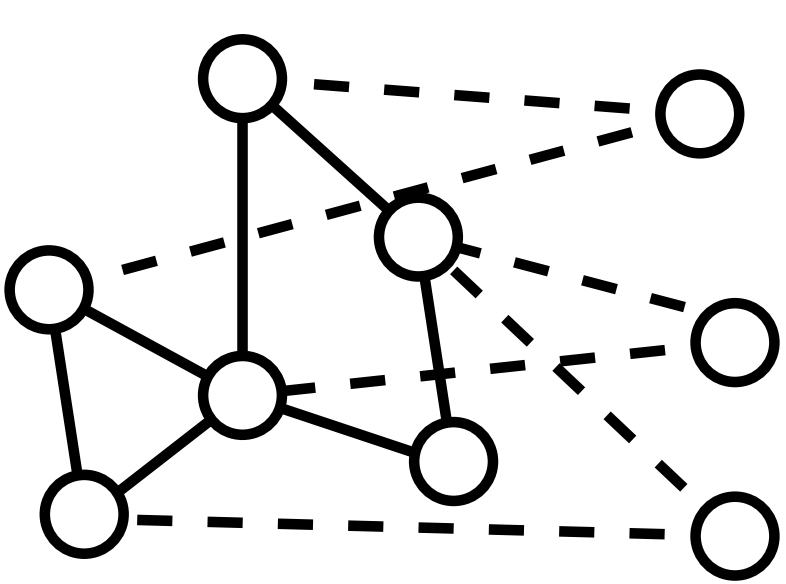
The idea (Bleakley et al., 2007)

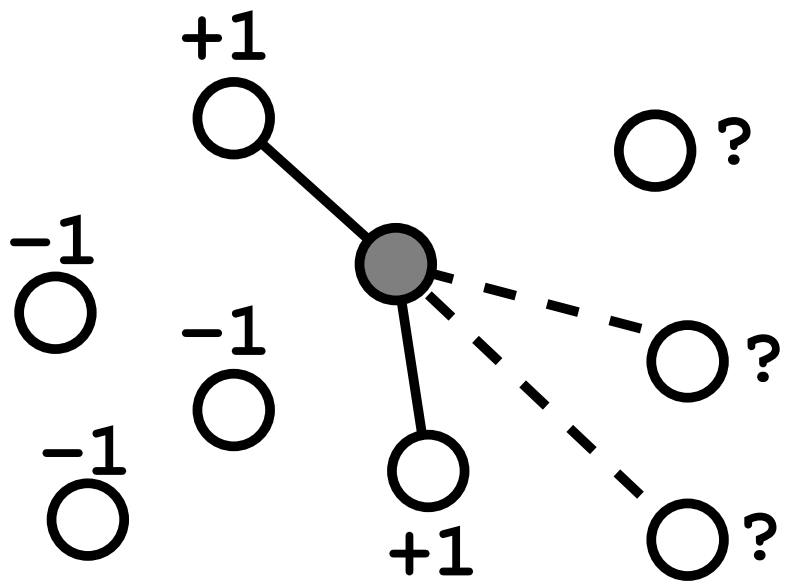
- Motivation: define specific models for each target node to discriminate between its neighbors and the others
- Treat each node independently from the other. Then combine predictions for ranking candidate edges.



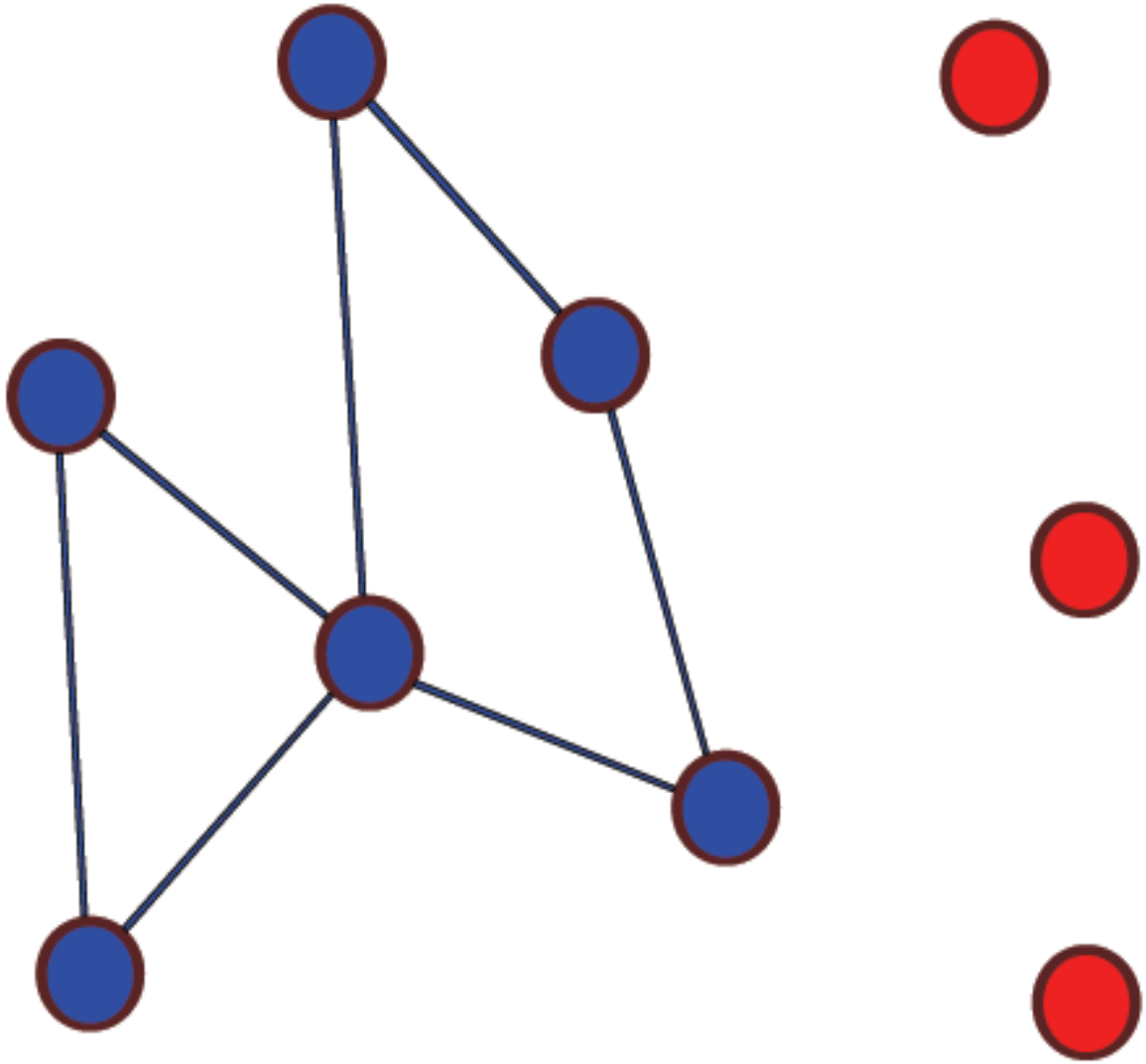
The idea (Bleakley et al., 2007)

- Motivation: define specific models for each target node to discriminate between its neighbors and the others
- Treat each node independently from the other. Then combine predictions for ranking candidate edges.

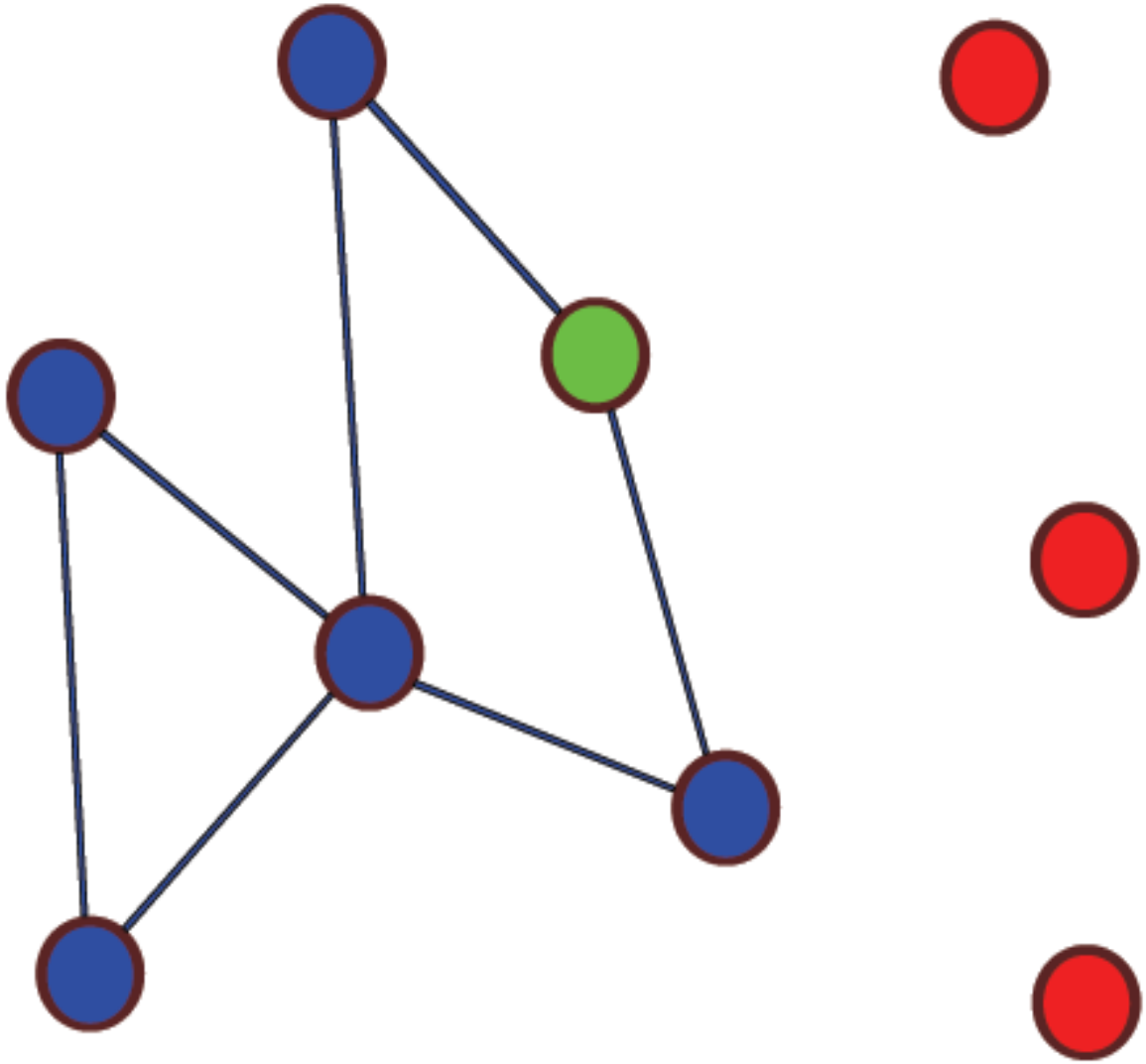




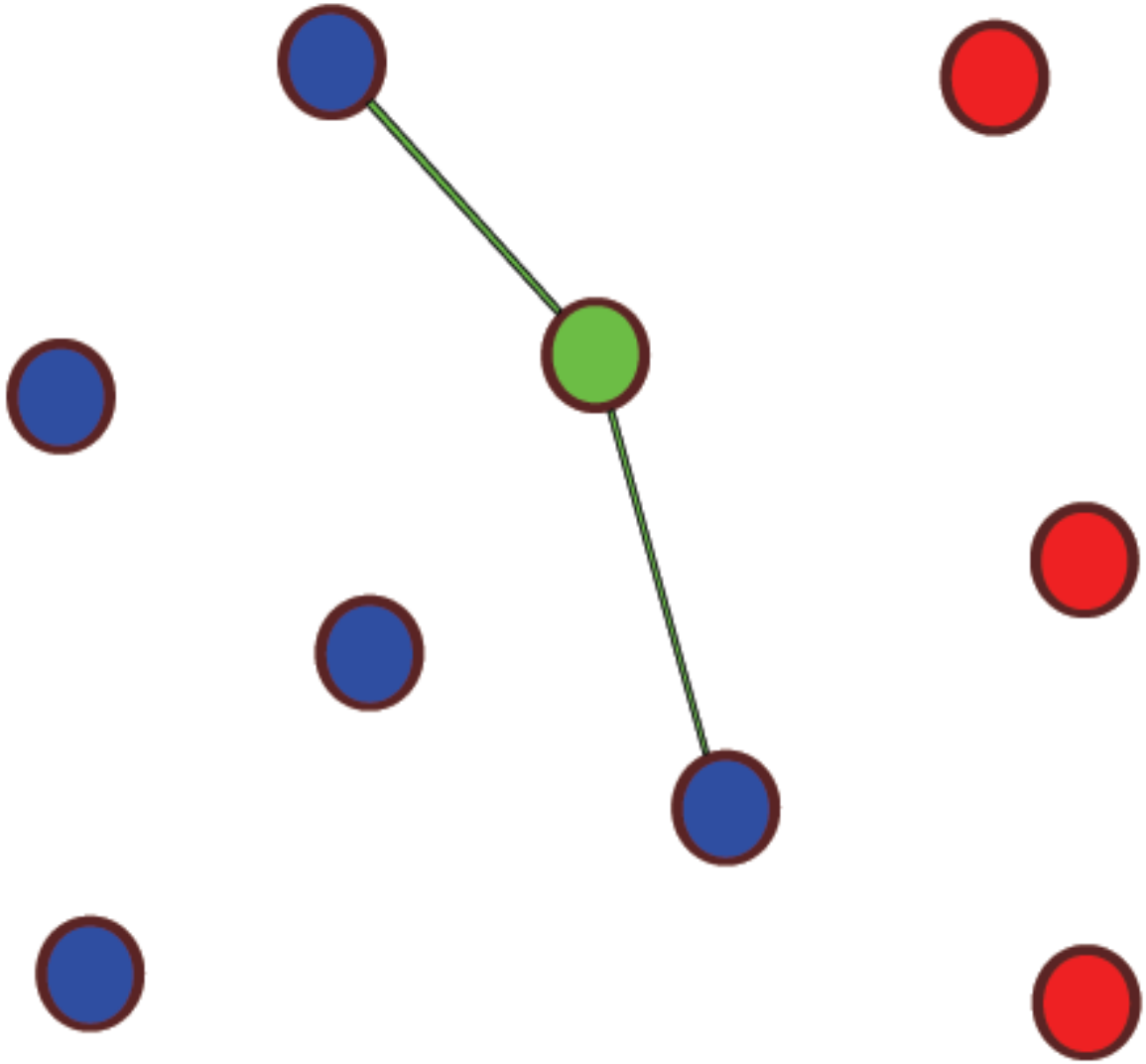
The LOCAL model



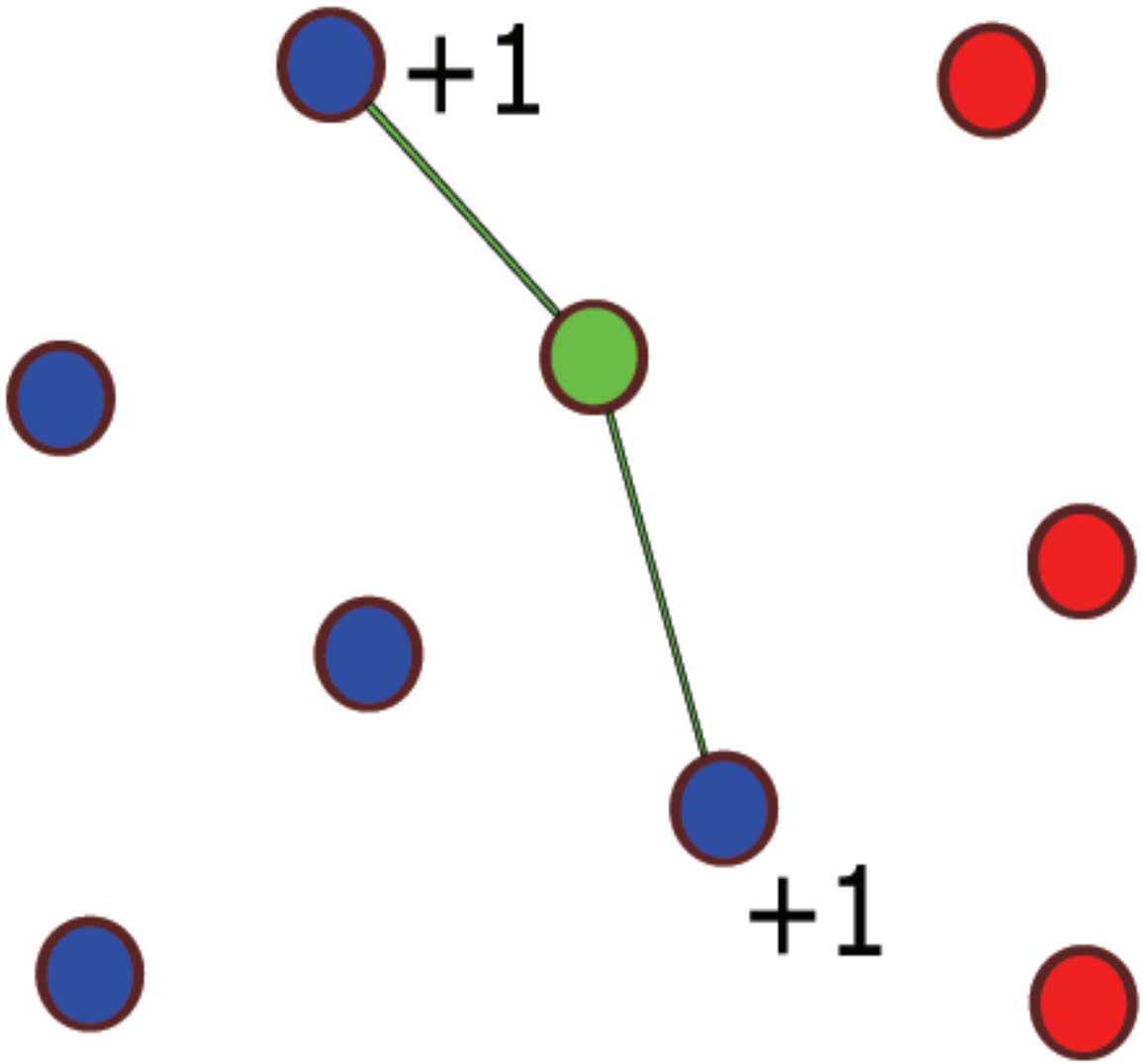
The LOCAL model



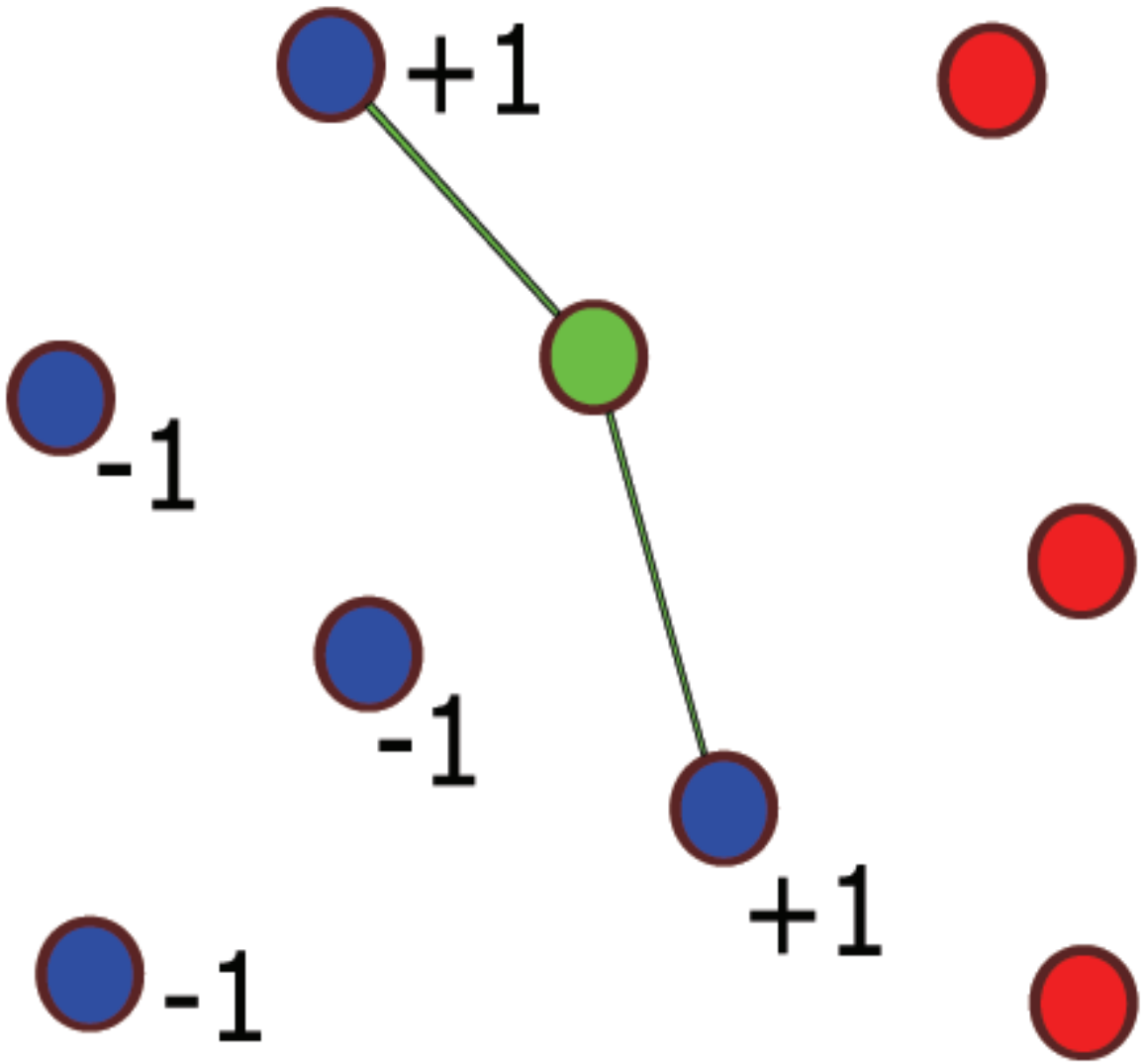
The LOCAL model



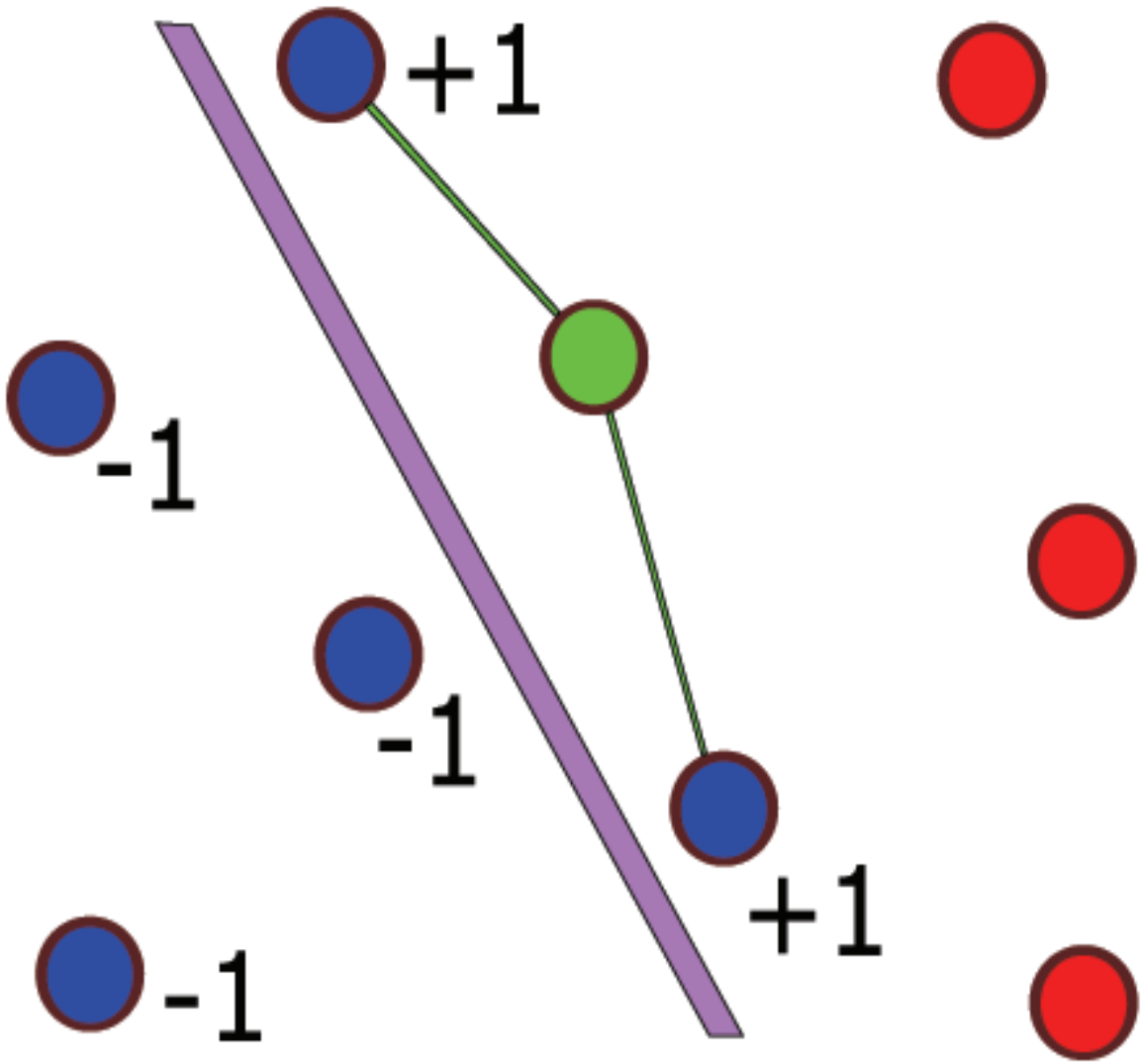
The LOCAL model



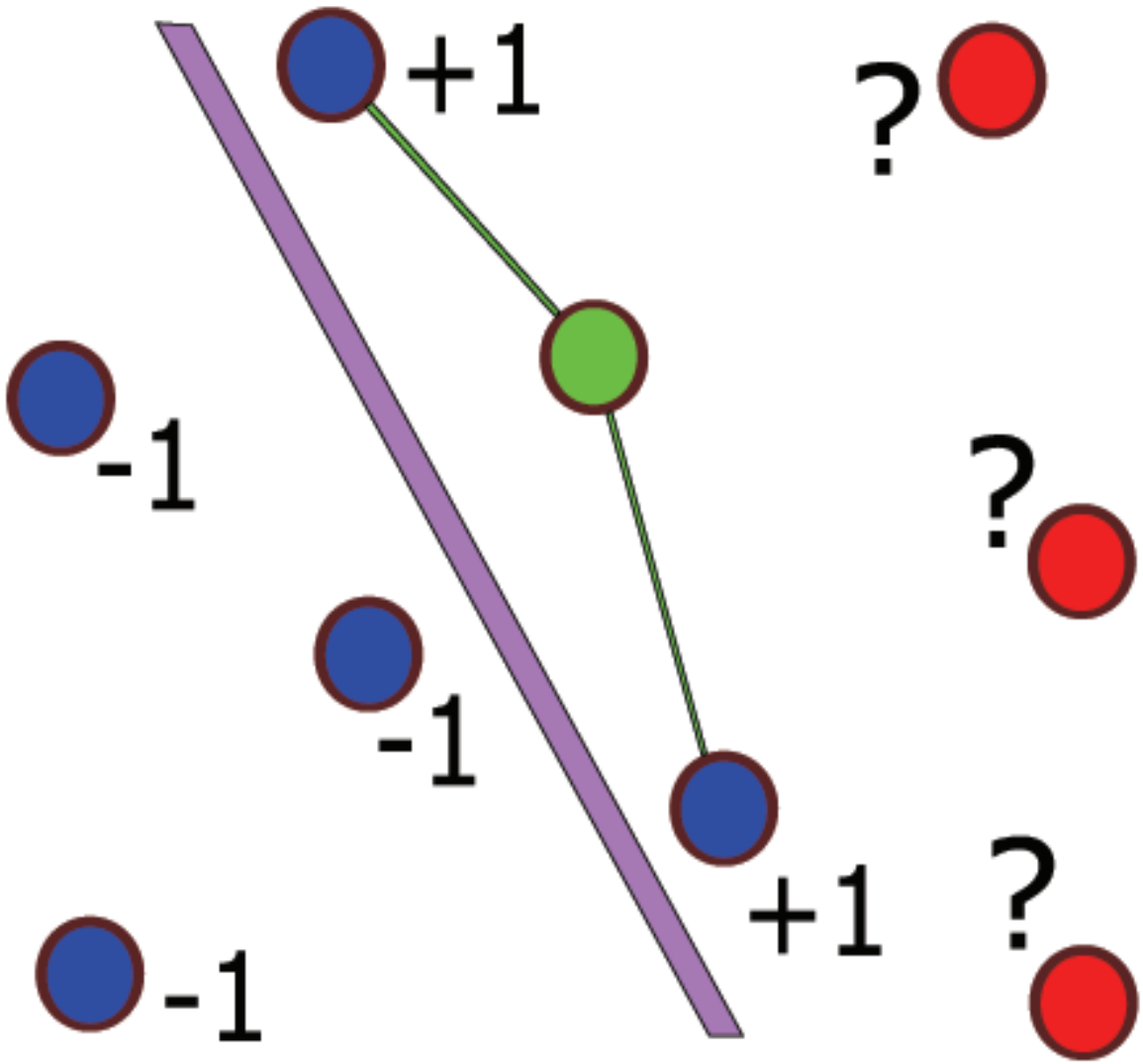
The LOCAL model



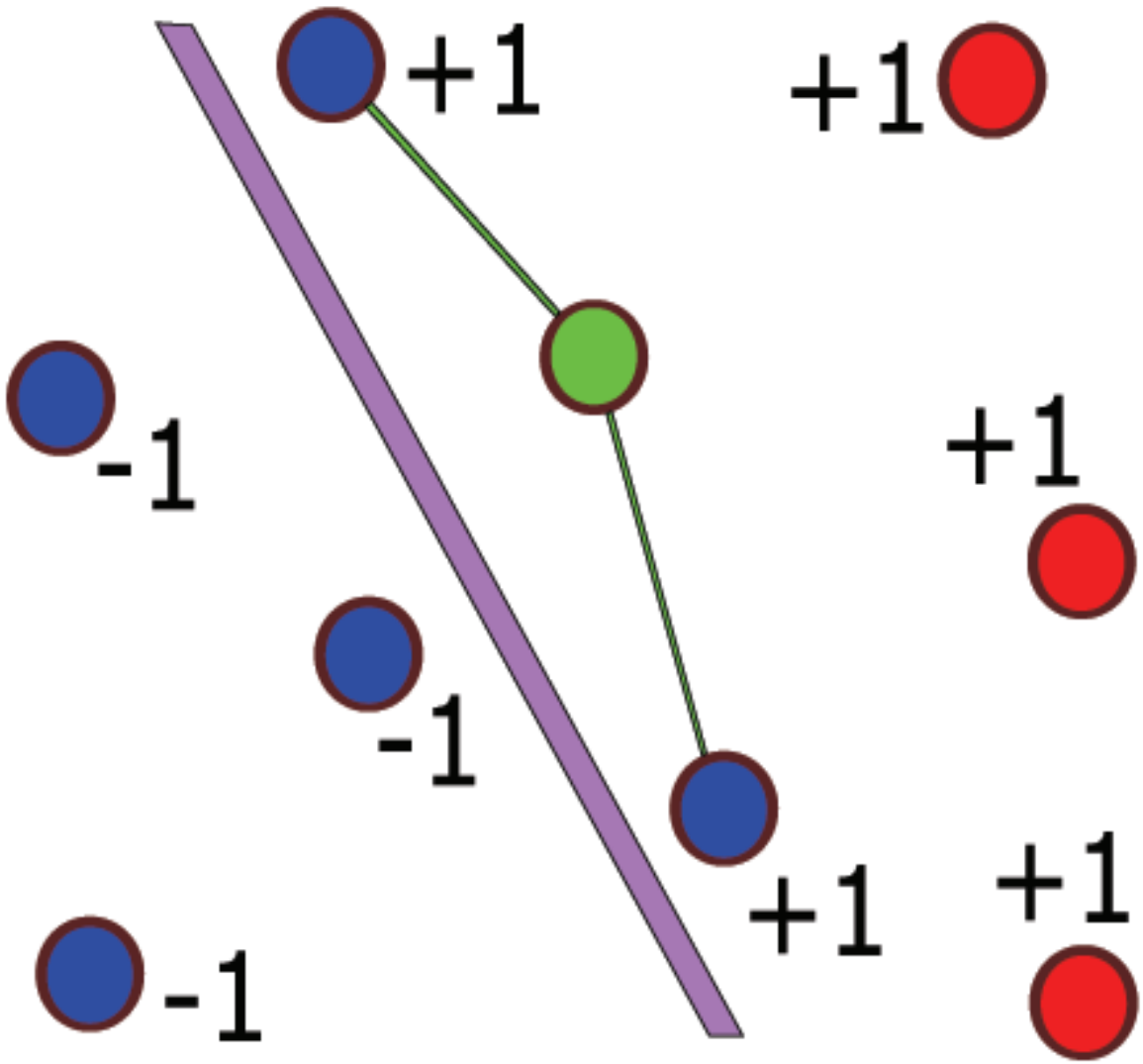
The LOCAL model



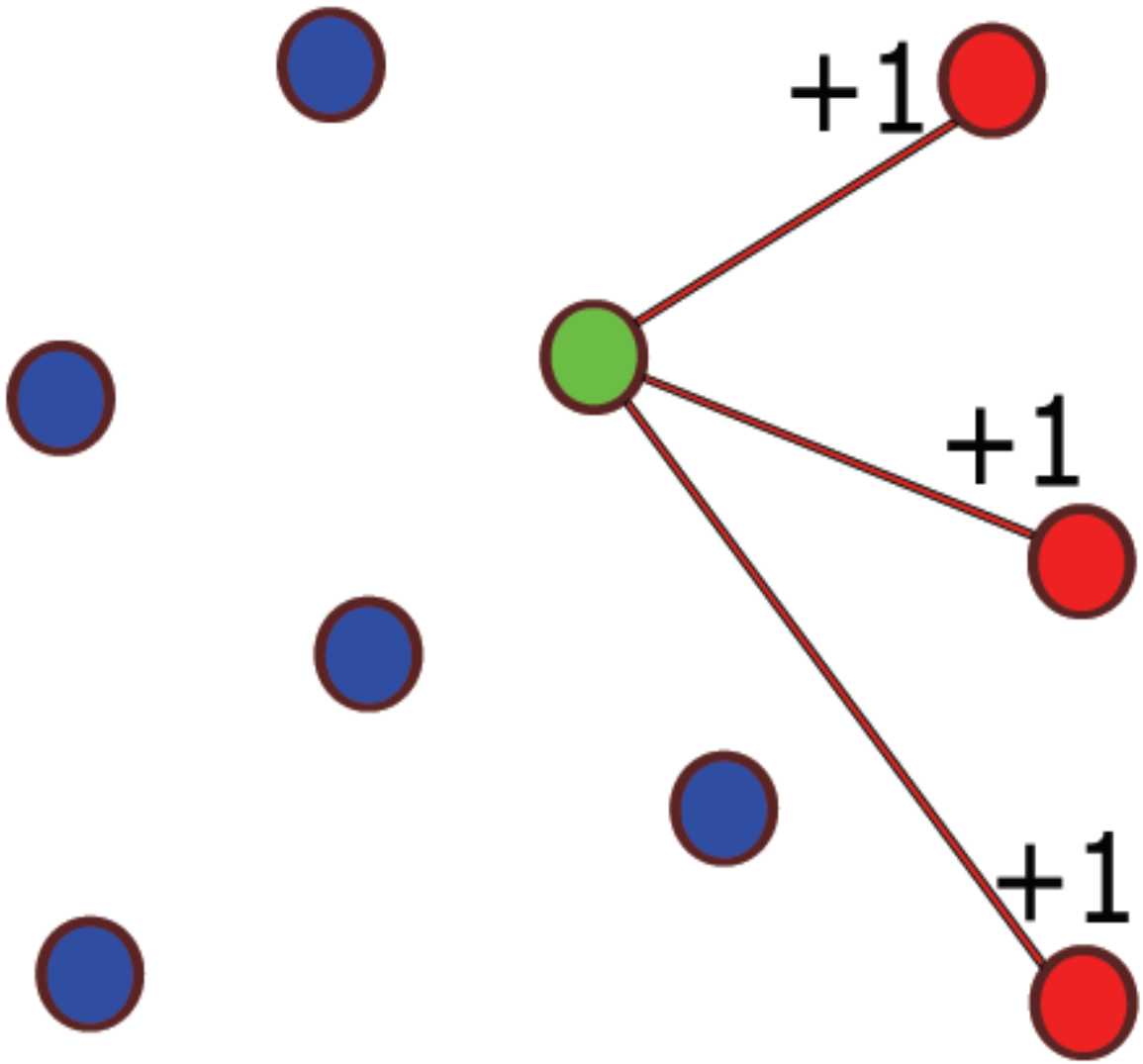
The LOCAL model



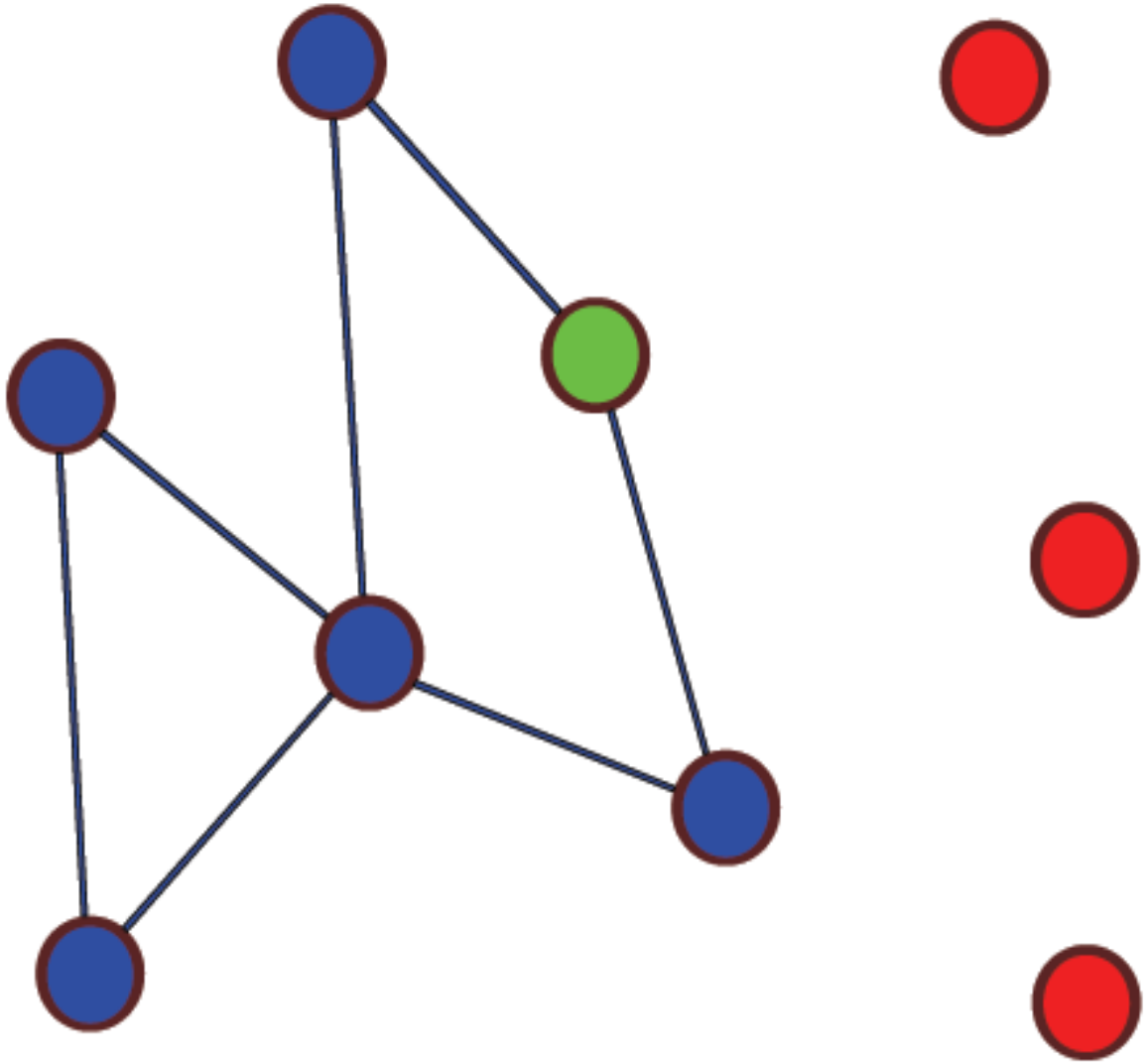
The LOCAL model



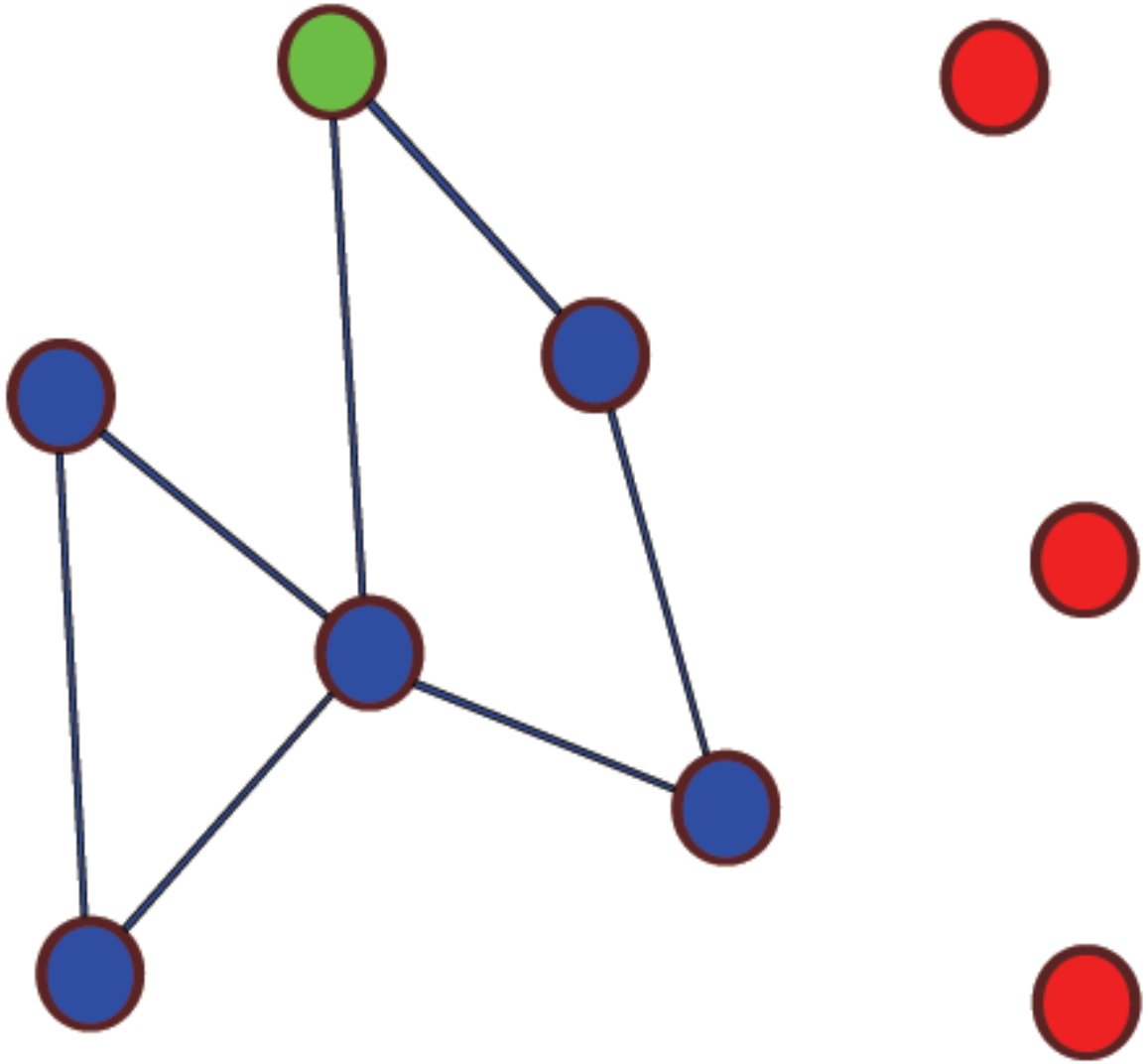
The LOCAL model



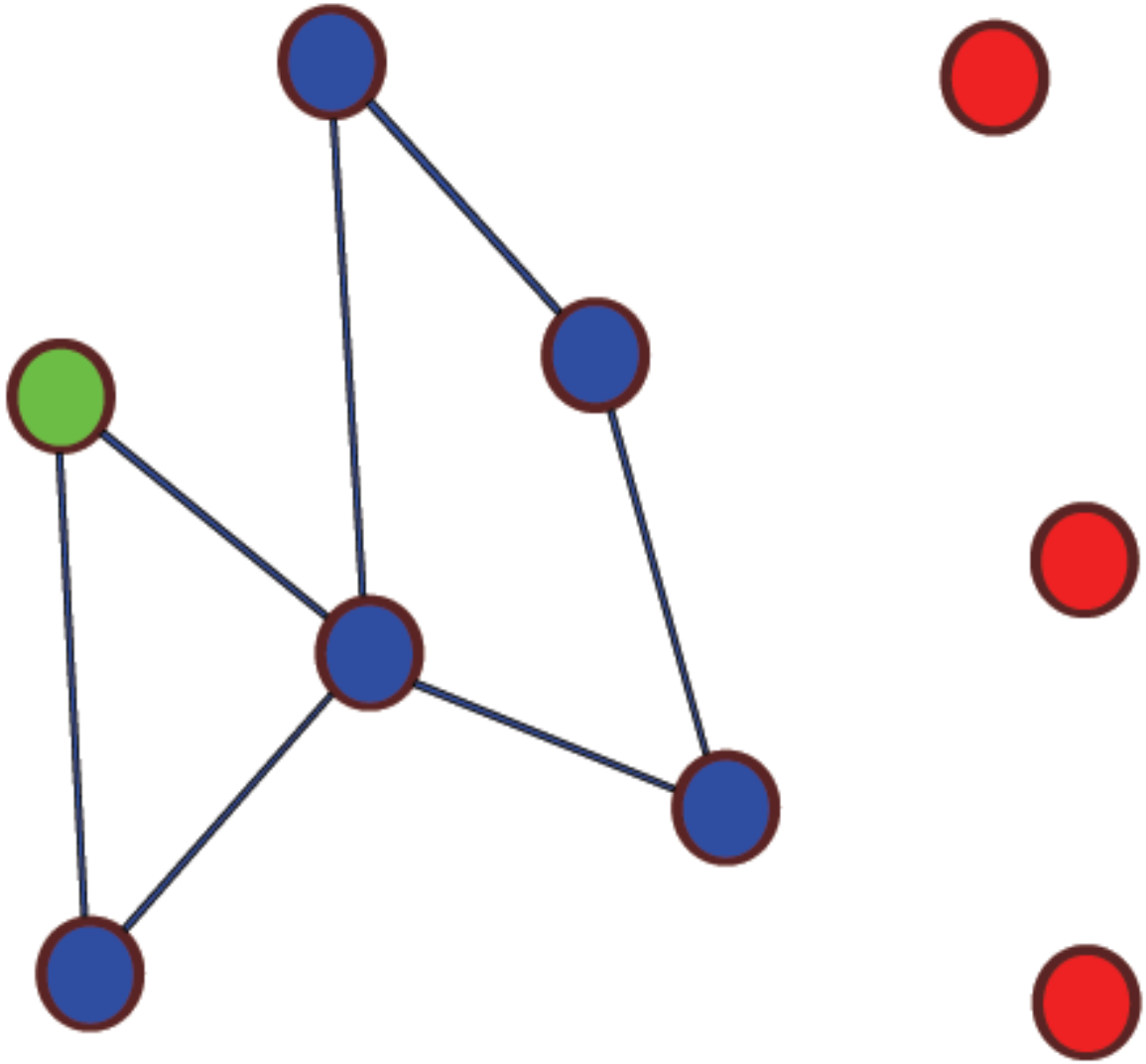
The LOCAL model



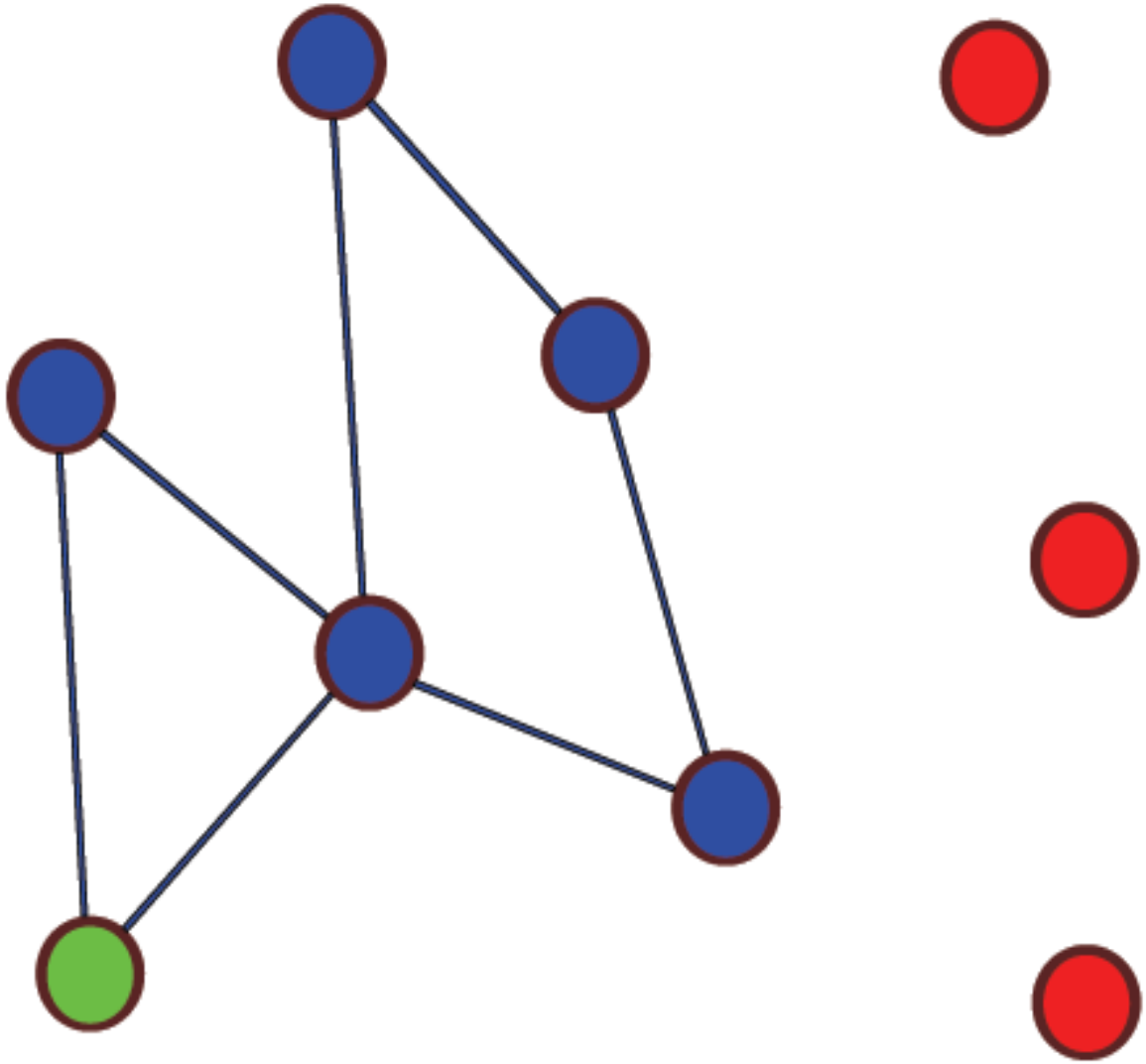
The LOCAL model



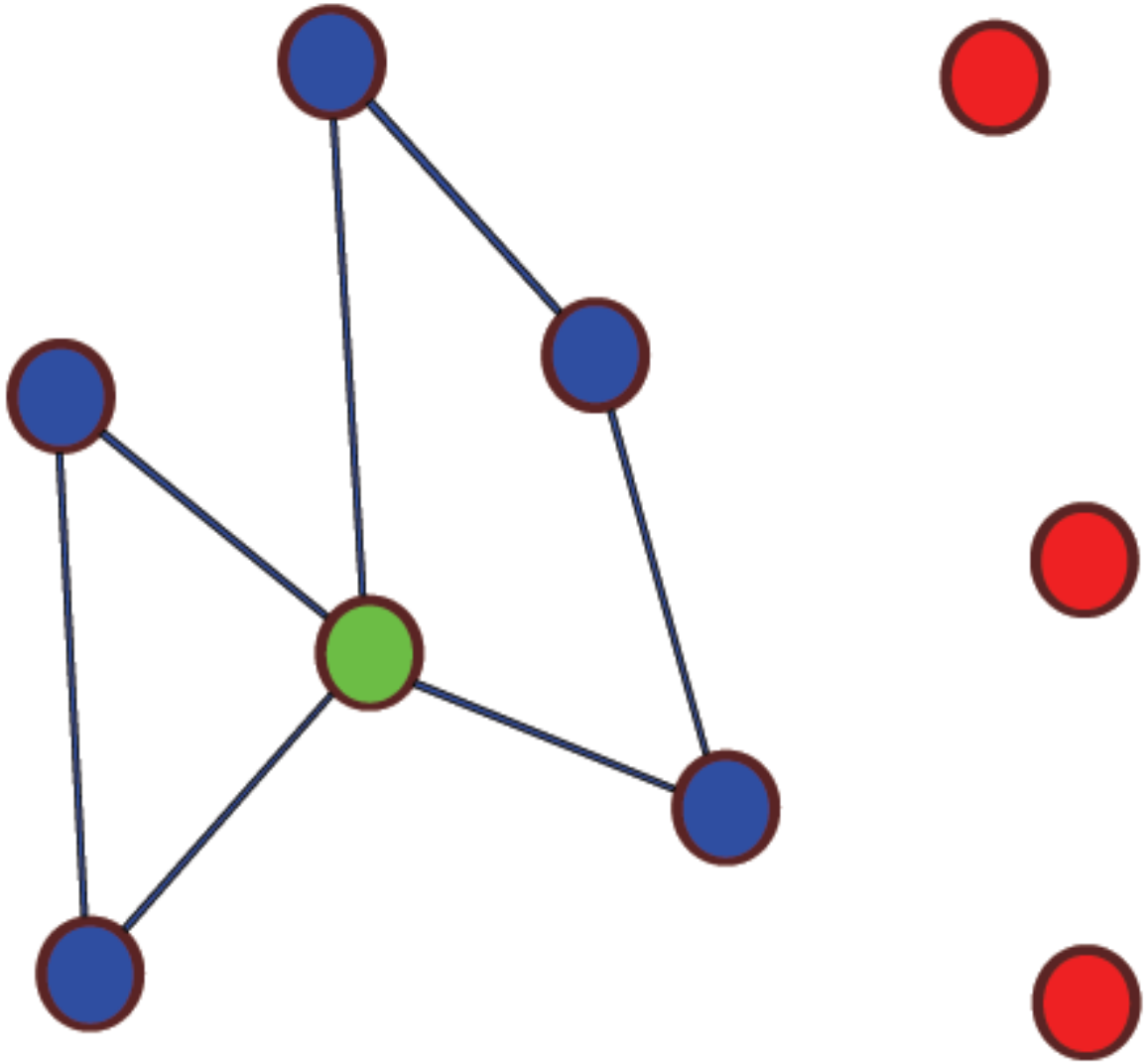
The LOCAL model



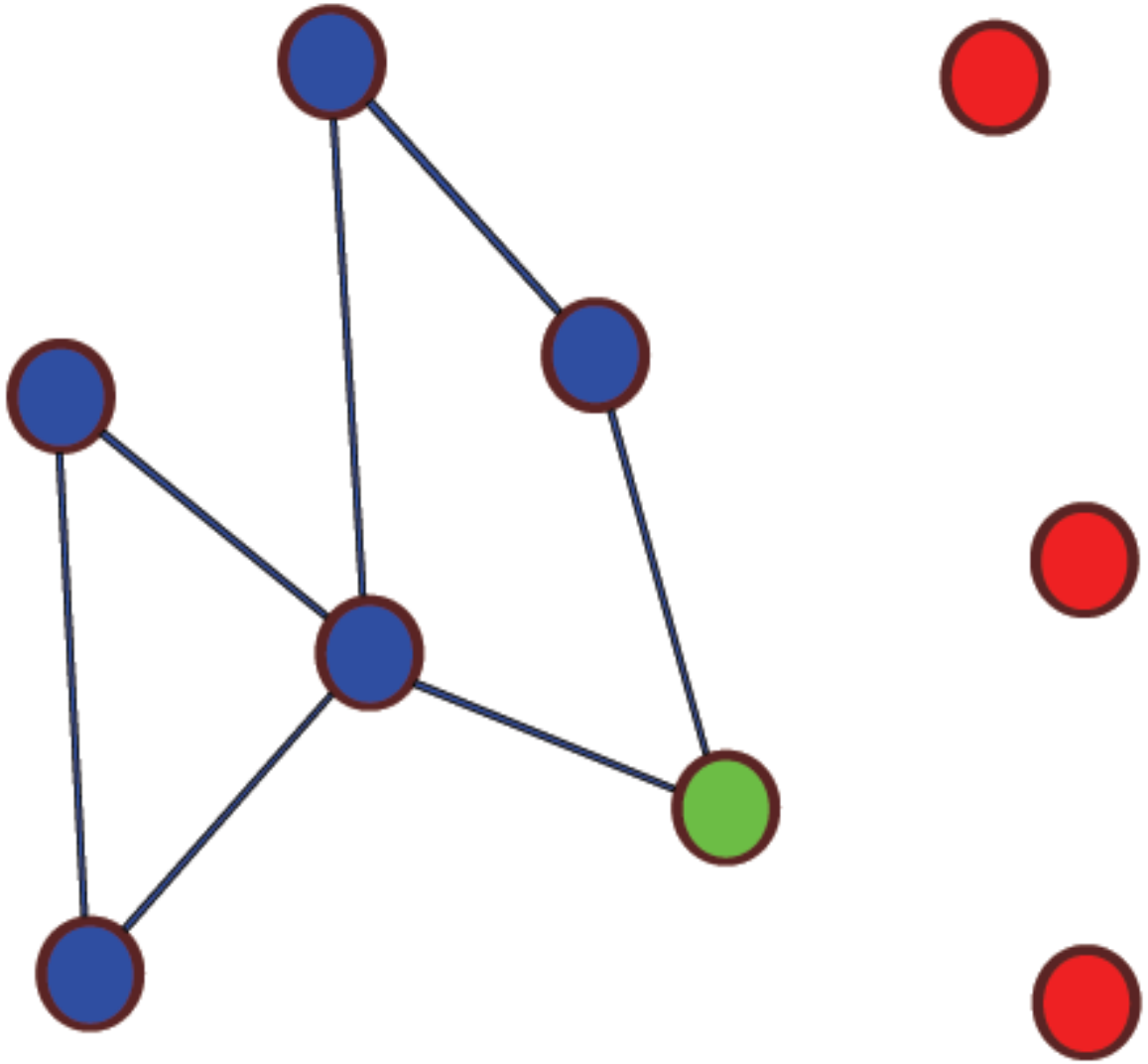
The LOCAL model



The LOCAL model



The LOCAL model



A few remarks

- In the case of unordered interactions, we need to symmetrize the prediction, typically by averaging the predictive scores of $A \to B$ and $B \to A$ to predict the interaction $\{A, B\}$
- Weak hypothesis:
  - if A is connected to B,
  - if C is similar to B,
  - then A is likely to be connected to C.
- Computationally: much faster to train N local models with N training points each, than to train 1 model with N^2 training points.
- Caveats:
  - each local model may have very few training points
  - no sharing of information between different local models

A few remarks

- In the case of unordered interactions, we need to symmetrize the prediction, typically by averaging the predictive scores of $A \to B$ and $B \to A$ to predict the interaction $\{A, B\}$
- Weak hypothesis:
  - if A is connected to B,
  - if C is similar to B,
  - then A is likely to be connected to C.
- Computationally: much faster to train N local models with N training points each, than to train 1 model with N^2 training points.
- Caveats:
  - each local model may have very few training points
  - no sharing of information between different local models

A few remarks

- In the case of unordered interactions, we need to symmetrize the prediction, typically by averaging the predictive scores of $A \to B$ and $B \to A$ to predict the interaction $\{A, B\}$
- Weak hypothesis:
  - if A is connected to B,
  - if C is similar to B,
  - then A is likely to be connected to C.
- Computationally: much faster to train N local models with N training points each, than to train 1 model with N^2 training points.
- Caveats:
  - each local model may have very few training points
  - no sharing of information between different local models

A few remarks

- In the case of unordered interactions, we need to symmetrize the prediction, typically by averaging the predictive scores of $A \to B$ and $B \to A$ to predict the interaction $\{A, B\}$
- Weak hypothesis:
  - if A is connected to B,
  - if C is similar to B,
  - then A is likely to be connected to C.
- Computationally: much faster to train N local models with N training points each, than to train 1 model with N^2 training points.
- Caveats:
  - each local model may have very few training points
  - no sharing of information between different local models

Outline

Motivation

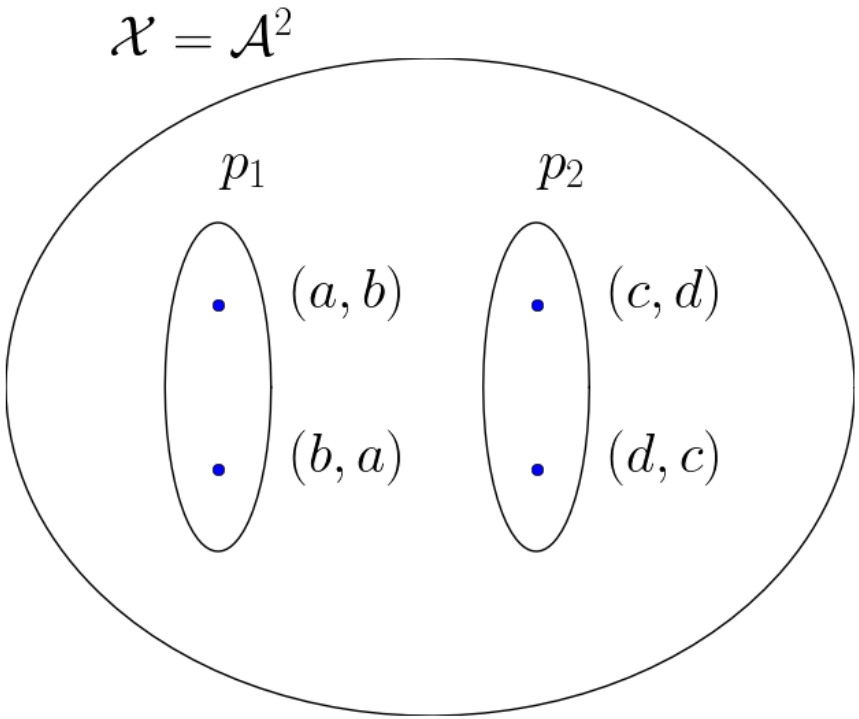
In the case of unordered pairs $\{A, B\}$, pairwise kernels such as the TPPK and local models look very different:

- Local models seem to over-emphasize the asymmetry of the relationships, but symmetrize the prediction *a posteriori*
- Pairwise kernels symmetrize the data *a priori* and learn in the space or unordered pairs

Can be clarify the links between these approaches, and perhaps interpolate between them?

Notations

- $\mathcal{A}$ the set of individual proteins, endowed with a kernel $K_{\mathcal{A}}$
- $\mathcal{X} = \mathcal{A}^2$ the set of ordered pairs of the form $x = (a, b)$ endowed with a kernel $K_{\mathcal{X}}$ (usually deduced from $K_{\mathcal{A}}$)
- $\mathcal{P}$ the set of unordered pairs of the form $p = \{(a, b), (b, a)\}$
- We want to learn over $\mathcal{P}$ from a set of labeled training pairs $(p_1, y_1), \ldots, (p_n, y_n) \in \mathcal{P} \times \{-1, 1\}$



Two strategies to learn over $\mathcal{P}$

Strategy 1: Inference over $\mathcal{P}$ with a pair kernel

1. Define a kernel $K_{\mathcal{P}}$ over $\mathcal{P}$ by convolution of $K_{\mathcal{X}}$:

$$K_{\mathcal{P}}(p, p') = \frac{1}{|p| \cdot |p'|} \sum_{x \in p, x' \in p'} K_{\mathcal{X}}(x, x') \,.$$

2. Train a classifier over $\mathcal{P}$ e.g., a SVM, using the kernel $K_{\mathcal{P}}$

Strategy 2: Inference over $\mathcal{X}$ with a pair duplication

1. Duplicate each training pair $p = \{a, b\}$ into 2 ordered paired
2. Train a classifier over $\mathcal{X}$, e.g., a SVM, using the kernel $K_{\mathcal{X}}$
3. The classifier over $\mathcal{P}$ is then the *a posteriori* average:

$$f_{\mathcal{P}}(p) = \frac{1}{|p|} \sum_{x \in p} f_{\mathcal{X}}(x)$$

Two strategies to learn over $\mathcal{P}$

Strategy 1: Inference over $\mathcal{P}$ with a pair kernel

1. Define a kernel $K_{\mathcal{P}}$ over $\mathcal{P}$ by convolution of $K_{\mathcal{X}}$:

$$K_{\mathcal{P}}(p, p') = \frac{1}{|p| \cdot |p'|} \sum_{x \in p, x' \in p'} K_{\mathcal{X}}(x, x') \,.$$

2. Train a classifier over $\mathcal{P}$ e.g., a SVM, using the kernel $K_{\mathcal{P}}$

Strategy 2: Inference over $\mathcal{X}$ with a pair duplication

1. Duplicate each training pair $p = \{a, b\}$ into 2 ordered paired
2. Train a classifier over $\mathcal{X}$, e.g., a SVM, using the kernel $K_{\mathcal{X}}$
3. The classifier over $\mathcal{P}$ is then the *a posteriori* average:

$$f_{\mathcal{P}}(p) = \frac{1}{|p|} \sum_{x \in p} f_{\mathcal{X}}(x)$$

The TPPK kernel

$$K_{TPPK}\left(\{a,b\},\{c,d\}\right) = K_{\mathcal{A}}(a,c)K_{\mathcal{A}}(b,d) + K_{\mathcal{A}}(a,d)K_{\mathcal{A}}(b,c)\,.$$

Theorem

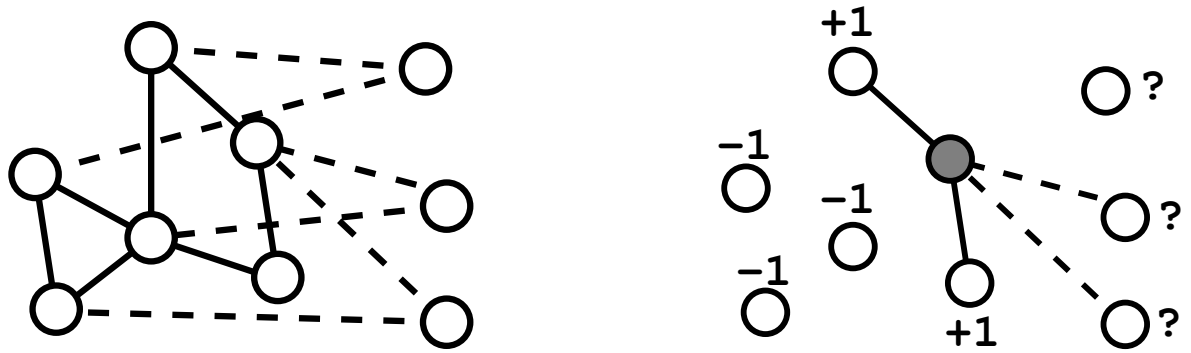
Let $\mathcal{X} = \mathcal{A}^2$ be endowed with the p.d. kernel:

$$K_{\mathcal{X}}\left((a,b),(c,d)\right) = 2K_{\mathcal{A}}(a,c)K_{\mathcal{A}}(b,d)\,. \tag{3}$$

Then the TPPK approach is equivalent to both Strategy 1 and Strategy 2.

Remarks: Equivalence with Strategy 1 is obvious, equivalence with Strategy 2 is not, see proof in Hue and V. (ICML 2010).

The local models



Theorem

Let $\mathcal{X} = \mathcal{A}^2$ be endowed with the p.d. kernel:

$$K_{\mathcal{X}}\left((a,b),(c,d)\right) = \delta(a,c)K_{\mathcal{A}}(b,d)\,,$$

where δ is the Kronecker kernel ($\delta(a,c) = 1$ if $a = c$, 0 otherwise). Then the local approach is equivalent to Strategy 2.

Remarks: Strategies 1 and 2 are not equivalent with this kernel. In general, they are equivalent up to a modification in the loss function of the learning algorithm, see details in Hue and V. (ICML 2010)..

Interpolation between local model and TPPK

	Strategy 1: pair kernel	Strategy 2: duplication
$K_{\mathcal{X}} = K_{\mathcal{A}} \otimes K_{\mathcal{A}}$	TPPK	TPPK
$K_{\mathcal{X}} = \delta \otimes K_{\mathcal{A}}$	new	Local model

Interpolation:

$$K_{\mathcal{X}} = ((1-\lambda)K_{\mathcal{A}} + \lambda\delta) \otimes K_{\mathcal{A}}$$

for $\lambda \in [0, 1]$

Interpolation between local model and TPPK

	Strategy 1: pair kernel	Strategy 2: duplication
$K_{\mathcal{X}} = K_{\mathcal{A}} \otimes K_{\mathcal{A}}$	TPPK	TPPK
$K_{\mathcal{X}} = \delta \otimes K_{\mathcal{A}}$	new	Local model

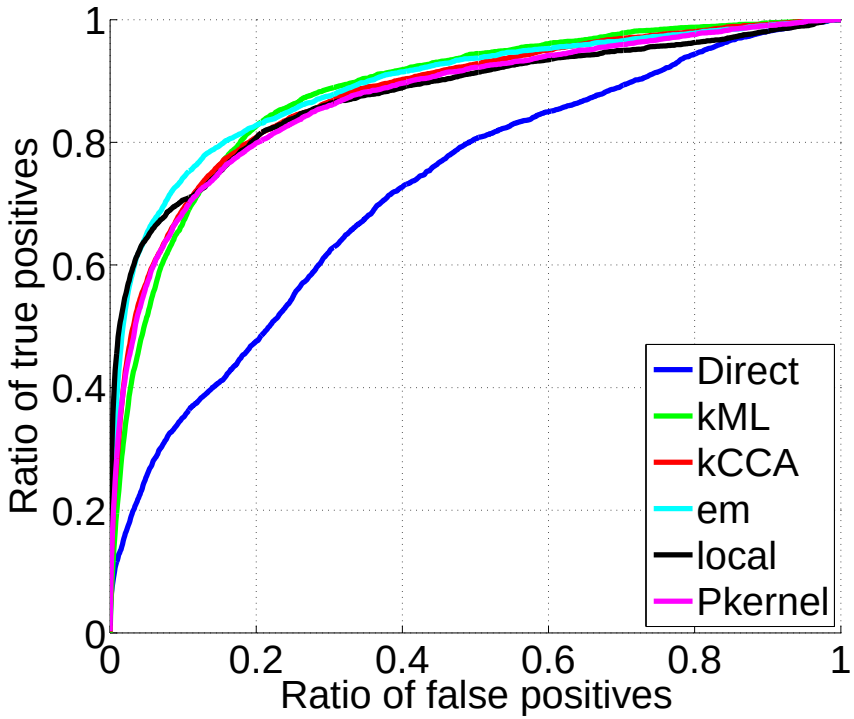
Interpolation:

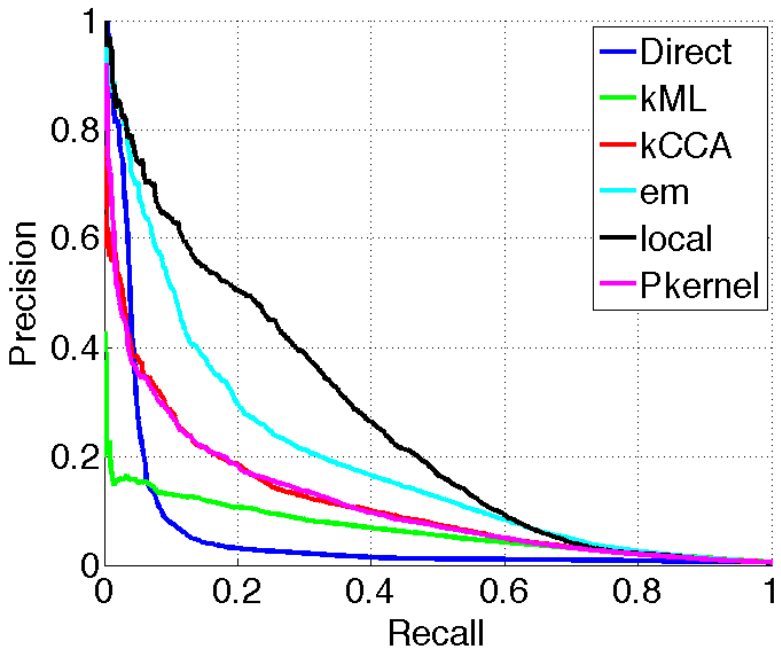
$$K_{\mathcal{X}} = ((1-\lambda)K_{\mathcal{A}} + \lambda\delta) \otimes K_{\mathcal{A}}$$

for $\lambda \in [0, 1]$

Outline

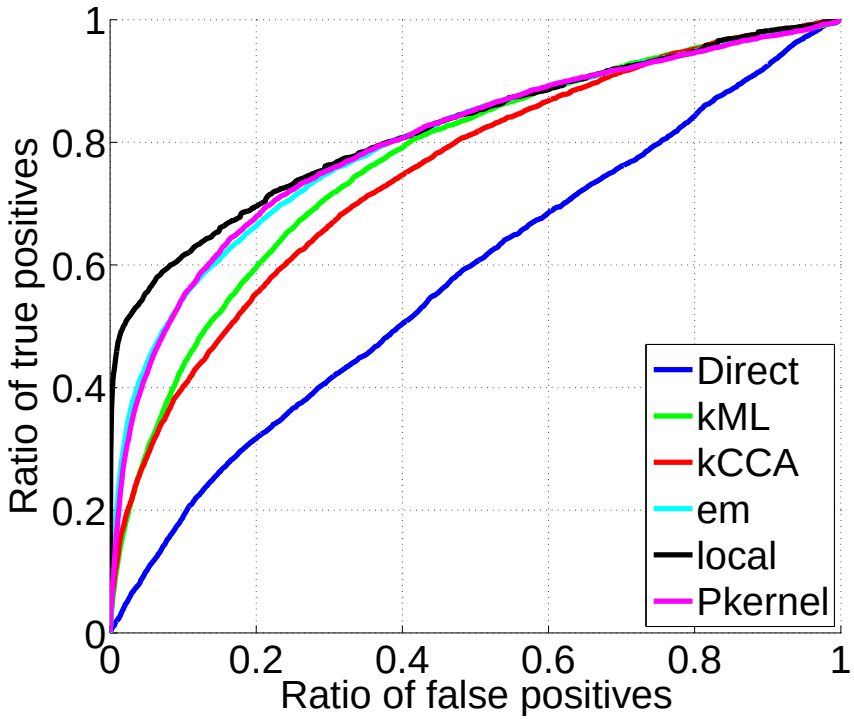
Results: protein-protein interaction (yeast)

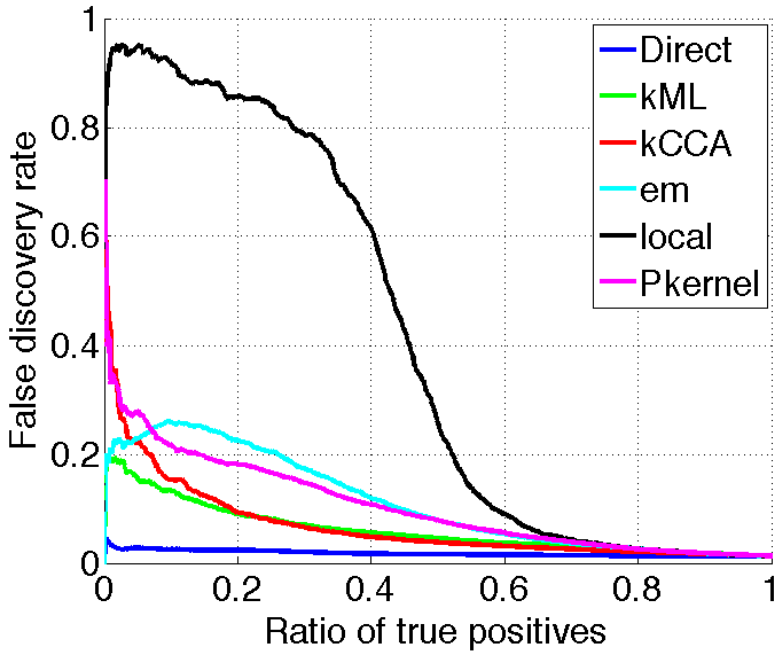




(from Bleakley et al., 2007)

Results: metabolic gene network (yeast)





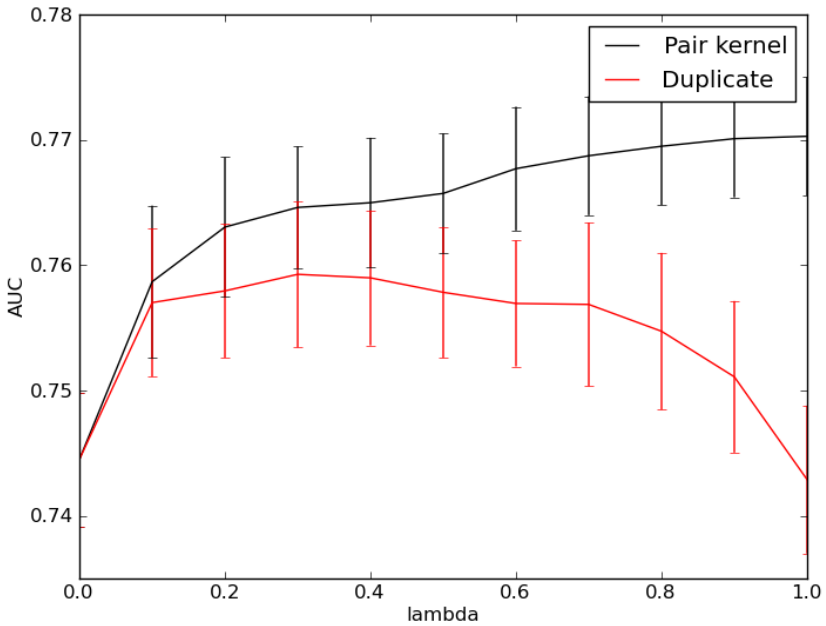
(from Bleakley et al., 2007)

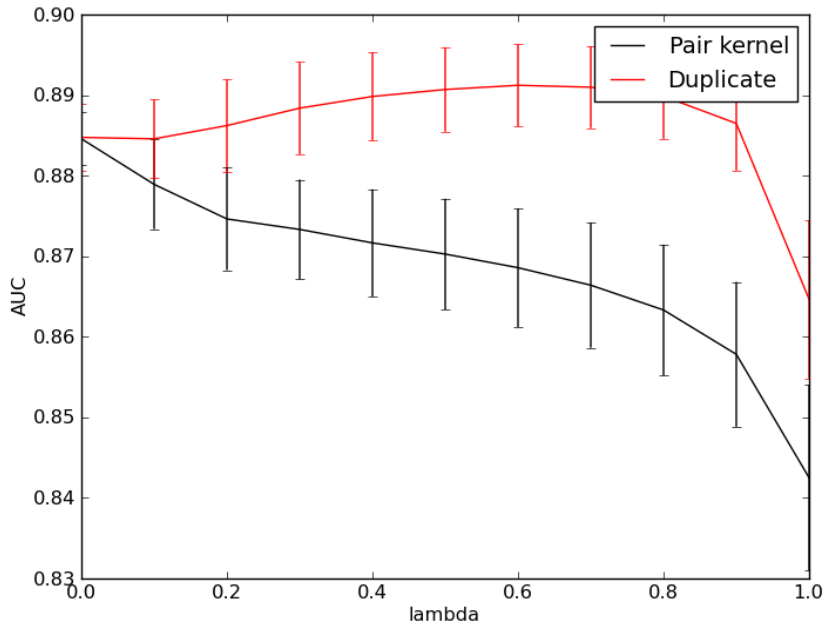
Interpolation kernel

Table: Strategy and kernel realizing the maximum mean AUC for nine metabolic and protein-protein interaction networks experiments, with the kernel K^λ for $\lambda \in [0, 1]$.

benchmark	best kernel
interaction, exp	Duplicate, $\lambda = 0.7$
interaction, loc	Pair kernel, $\lambda = 0.6$
interaction, phy	Duplicate, $\lambda = 0.8$
interaction, y2h	Duplicate / Pair kernel, $\lambda = 0$
interaction, integrated	Duplicate / Pair kernel, $\lambda = 0$
metabolic, exp	Pair kernel, $\lambda = 0.6$
metabolic, loc	Pair kernel, $\lambda = 1$
metabolic, phy	Pair kernel, $\lambda = 0.6$
metabolic, integrated	Duplicate / Pair kernel, $\lambda = 0$

Interpolation kernel





Metabolic networks with localization data (left); PPI network with expression data (right)

Applications: missing enzyme prediction

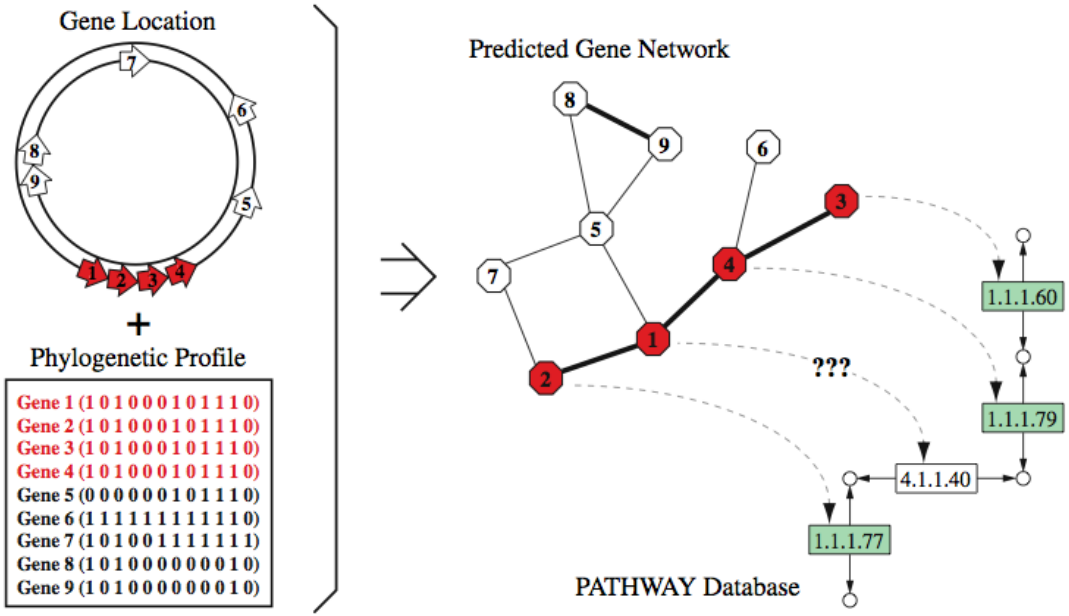
the FEBS Journal

Prediction of missing enzyme genes in a bacterial metabolic network

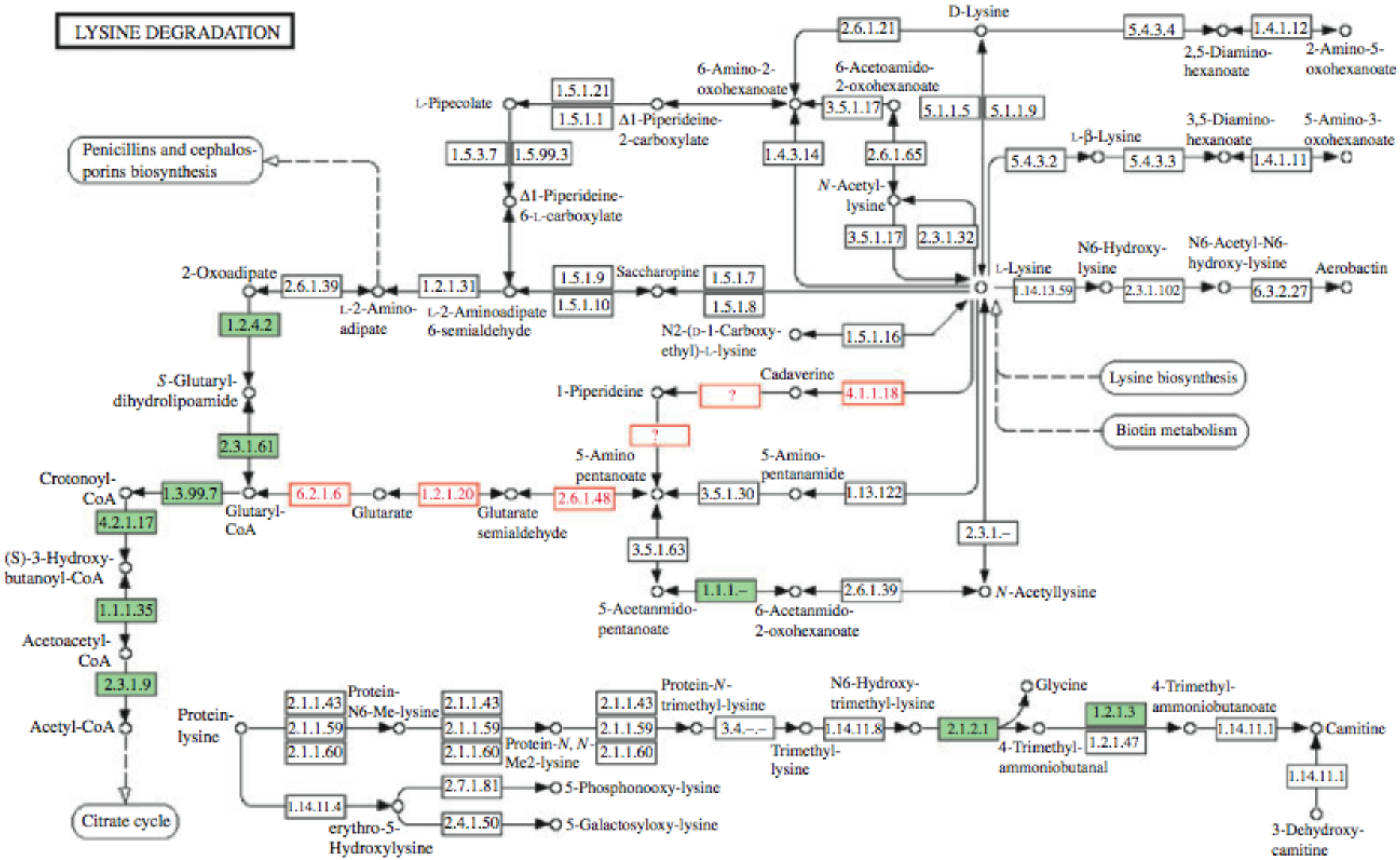
Reconstruction of the lysine-degradation pathway of *Pseudomonas aeruginosa*

Yoshihiro Yamanishi[1], Hisaaki Mihara[2], Motoharu Osaki[2], Hisashi Muramatsu[3], Nobuyoshi Esaki[2], Tetsuya Sato[1], Yoshiyuki Hizukuri[1], Susumu Goto[1] and Minoru Kanehisa[1]

1 Bioinformatics Center, Institute for Chemical Research, Kyoto University, Japan
2 Division of Environmental Chemistry, Institute for Chemical Research, Kyoto University, Japan
3 Department of Biology, Graduate School of Science, Osaka University, Japan



Applications: missing enzyme prediction



Applications: missing enzyme prediction

900 DOI 10.1002/pmic.200600862 *Proteomics* 2007, *7*, 900–909

RESEARCH ARTICLE

Prediction of nitrogen metabolism-related genes in *Anabaena* by kernel-based network analysis

Shinobu Okamoto[1], Yoshihiro Yamanishi[1], Shigeki Ehira[2], Shuichi Kawashima[3], Koichiro Tonomura[1]** and Minoru Kanehisa[1]*

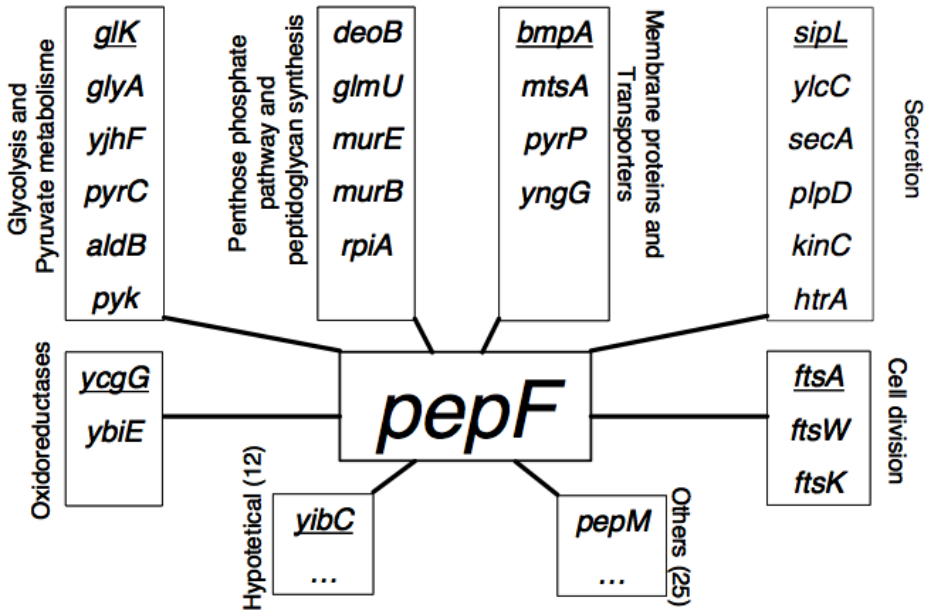
[1] Bioinformatics Center, Institute for Chemical Research, Kyoto University, Uji, Japan
[2] Department of Biochemistry and Molecular Biology, Faculty of Science, Saitama University, Saitama, Japan
[3] Human Genome Center, Institute of Medical Science, University of Tokyo, Meguro, Japan

Applications: function annotation

Determination of the role of the bacterial peptidase PepF by statistical inference and further experimental validation

Liliana LOPEZ KLEINE[1,2], Alain TRUBUIL[1], Véronique MONNET[2]

[1]Unité de Mathématiques et Informatiques Appliquées. INRA Jouy en Josas 78352, France.
[2]Unité de Biochimie Bactérienne. INRA Jouy en Josas 78352, France.



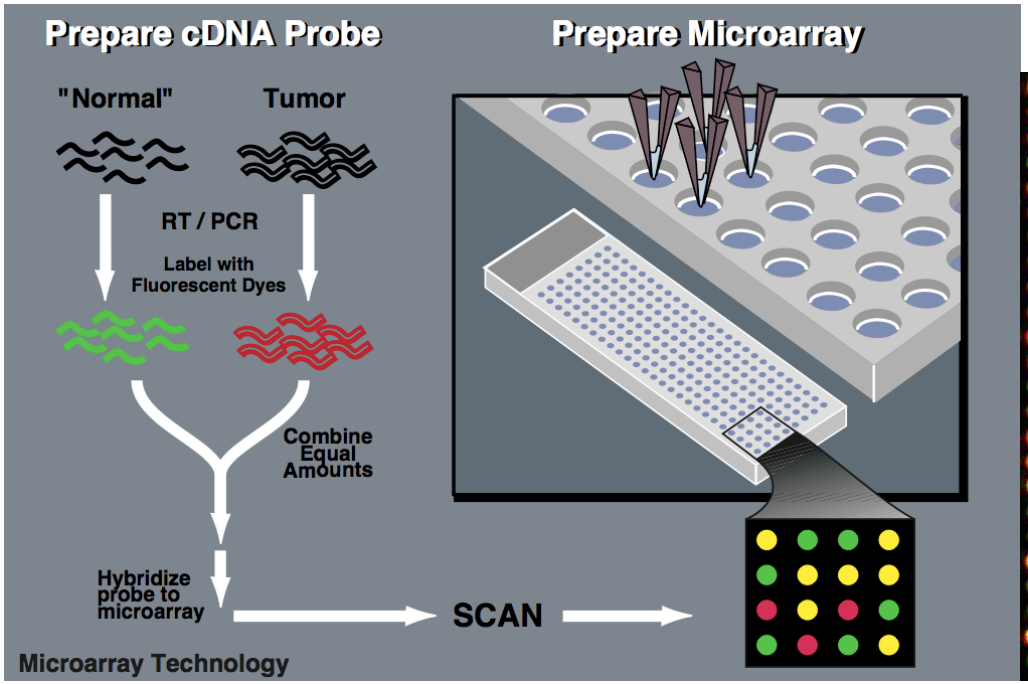
Conclusion

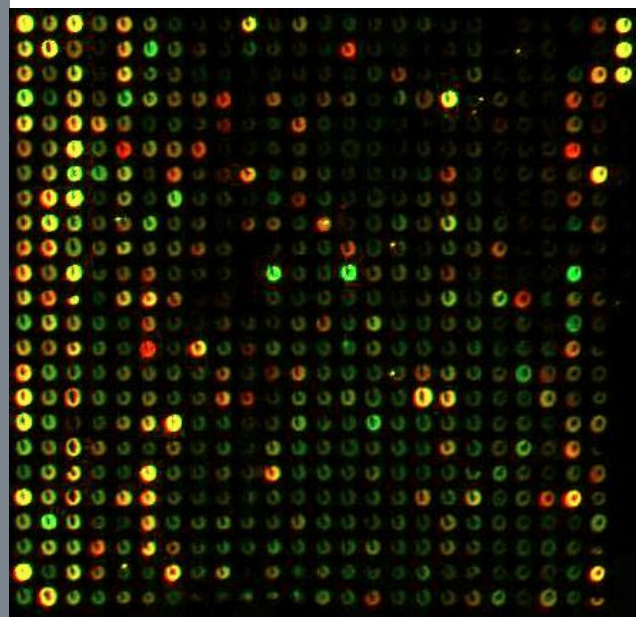
- When the network is known in part, supervised methods are more adapted than unsupervised ones.
- A variety of methods have been investigated recently (metric learning, matrix completion, pattern recognition).
  - work for any network
  - work with any data
  - can integrate heterogeneous data, which strongly improves performance
- Promising topic: infer edges simultaneously with global constraints on the graph?

Outline

Outline

Tissue profiling with DNA chips

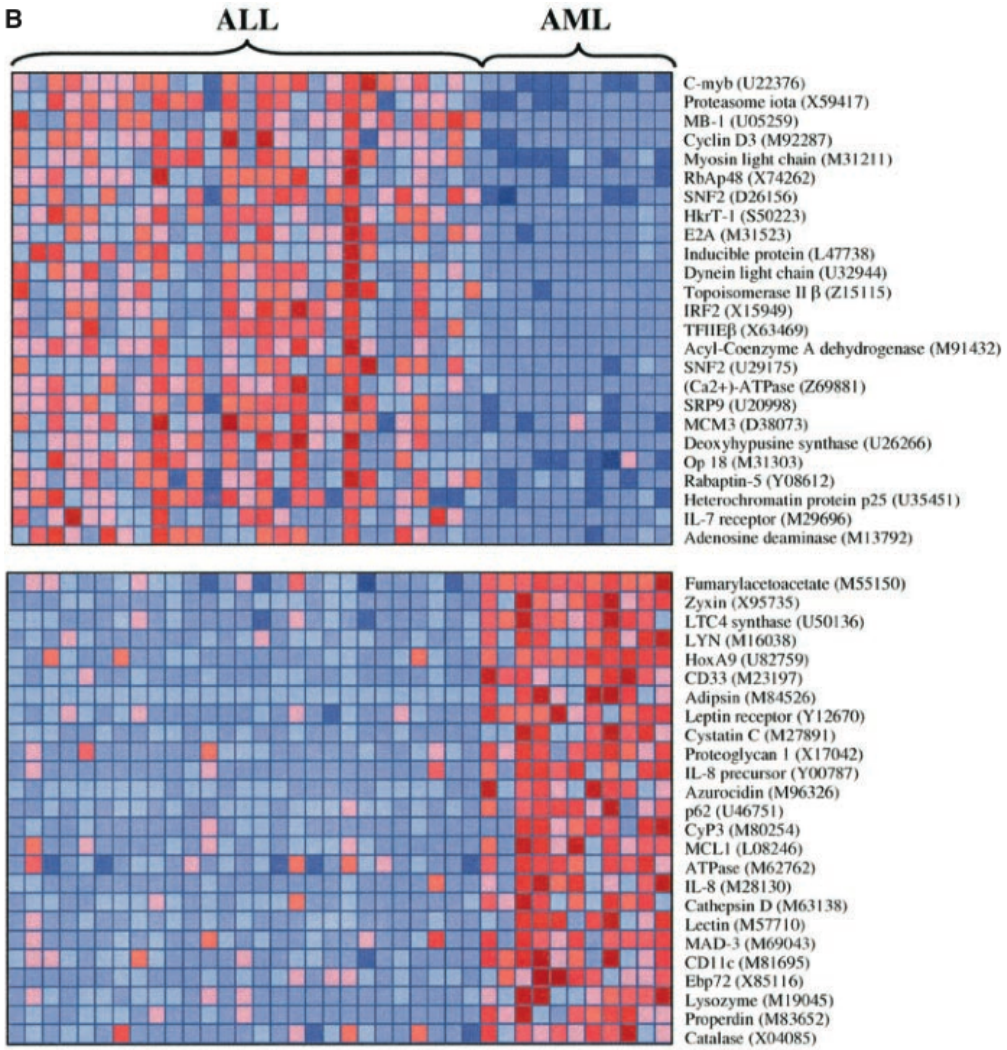




Data

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

Tissue classification from microarray data



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:

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

  that is positive for one class, negative for the other
- Interpretation: the weight β_i quantifies the influence of gene i for the classification

Linear classifiers estimation

Empirical risk minimization

Estimate the weights β_i by minimizing an empirical error on the training set:

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

where $l(y, f(x))$ is a loss function.

Pitfalls

- Statistics does not apply (?): 100 samples in 10^5 dimensions!
- It is necessary to reduce the complexity of the problem with prior knowledge.

Linear classifiers estimation

Empirical risk minimization

Estimate the weights β_i by minimizing an empirical error on the training set:

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

where $l(y, f(x))$ is a loss function.

Pitfalls

- Statistics does not apply (?): 100 samples in 10^5 dimensions!
- It is necessary to reduce the complexity of the problem with prior knowledge.

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

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

(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 enough for classification. Interpretation is then about the selected genes. Examples:

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

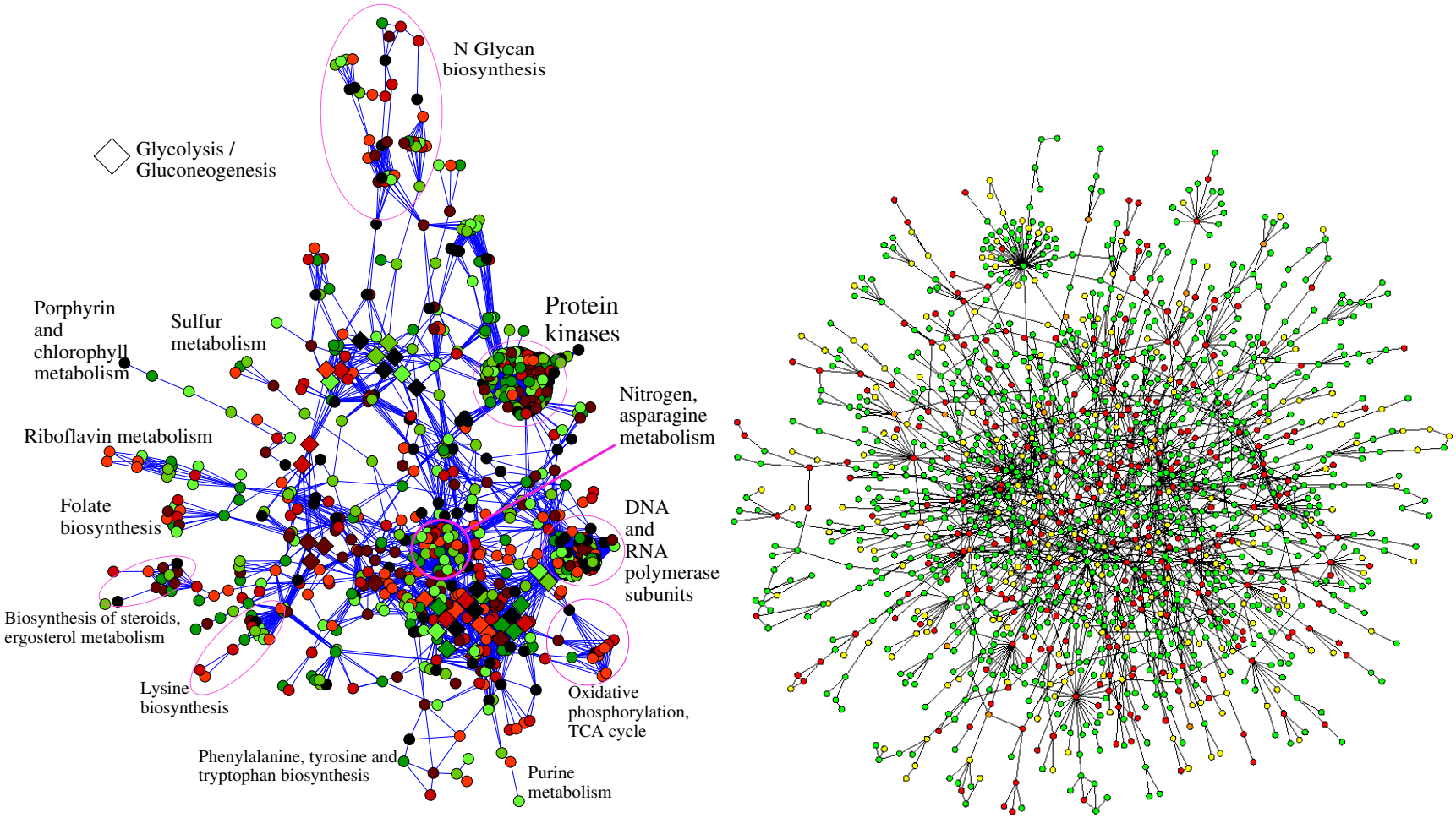
Pros

- Good performance in classification
- Biomarker selection
- Interpretability

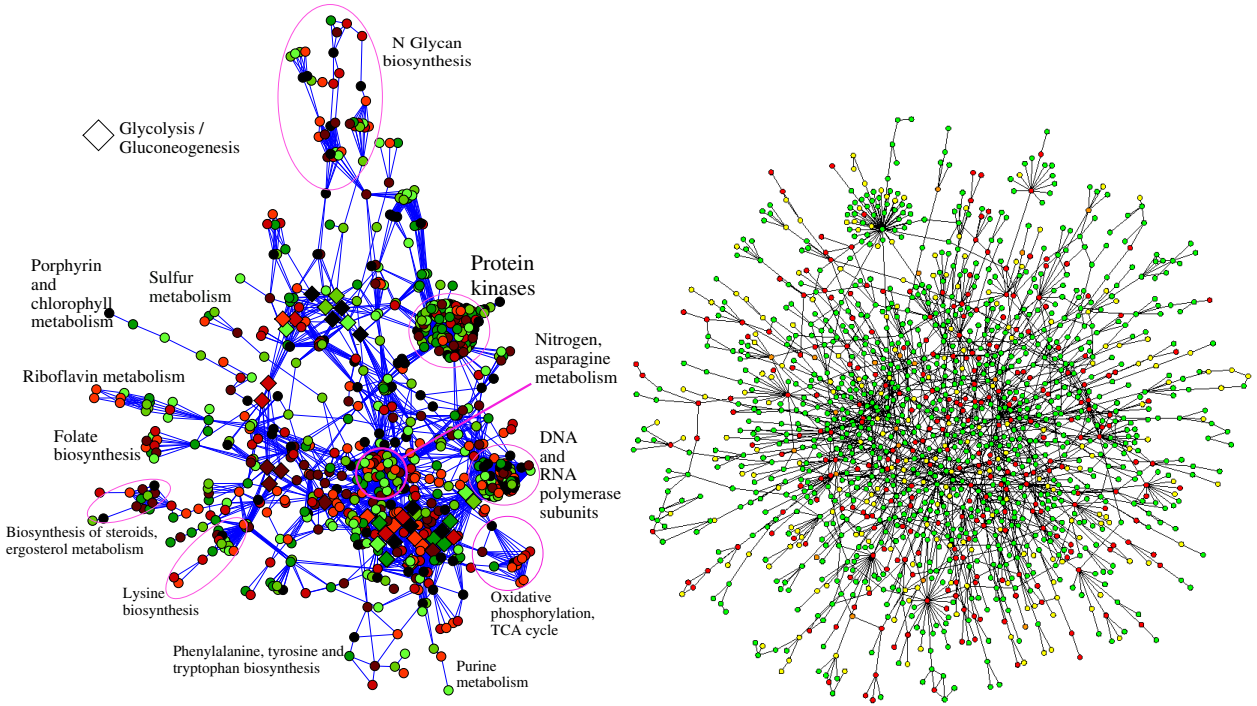
Cons

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

Gene networks



Gene networks



Assuming you give me a reliable gene network as prior knowledge, can it be helpful for the classification problem?

Outline

Gene network interpretation

Motivation

- Basic biological functions usually involve the coordinated action of several proteins:
  - Formation of protein complexes
  - Activation of metabolic, signalling or regulatory pathways
- Many pathways and protein-protein interactions are already known
- Hypothesis: the weights of the classifier should be "coherent" with respect to this prior knowledge

Reference

BMC Bioinformatics

BioMed Central

Research article

Open Access

Classification of microarray data using gene networks

Franck Rapaport*[1,2], Andrei Zinovyev[1], Marie Dutreix[3], Emmanuel Barillot[1] and Jean-Philippe Vert[2]

Address: [1]Institut Curie, Service de Bioinformatique, 26 rue d'Ulm, F-75248 Paris Cedex 05, France, [2]Ecole des Mines de Paris, Centre for Computational Biology, 35 rue Saint-Honoré, 77300 Fontainebleau, France and [3]Institut Curie, CNRS-UMR 2027, Bâtiment 110, Centre Universitaire, F-91405 Orsay, France

Email: Franck Rapaport* - franck.rapaport@curie.fr; Andrei Zinovyev - andrei.zinovyev@curie.fr; Marie Dutreix - marie.dutreix@curie.fr; Emmanuel Barillot - emmanuel.barillot@curie.fr; Jean-Philippe Vert - jean-philippe.vert@mines.org

* Corresponding author

Published: 1 February 2007

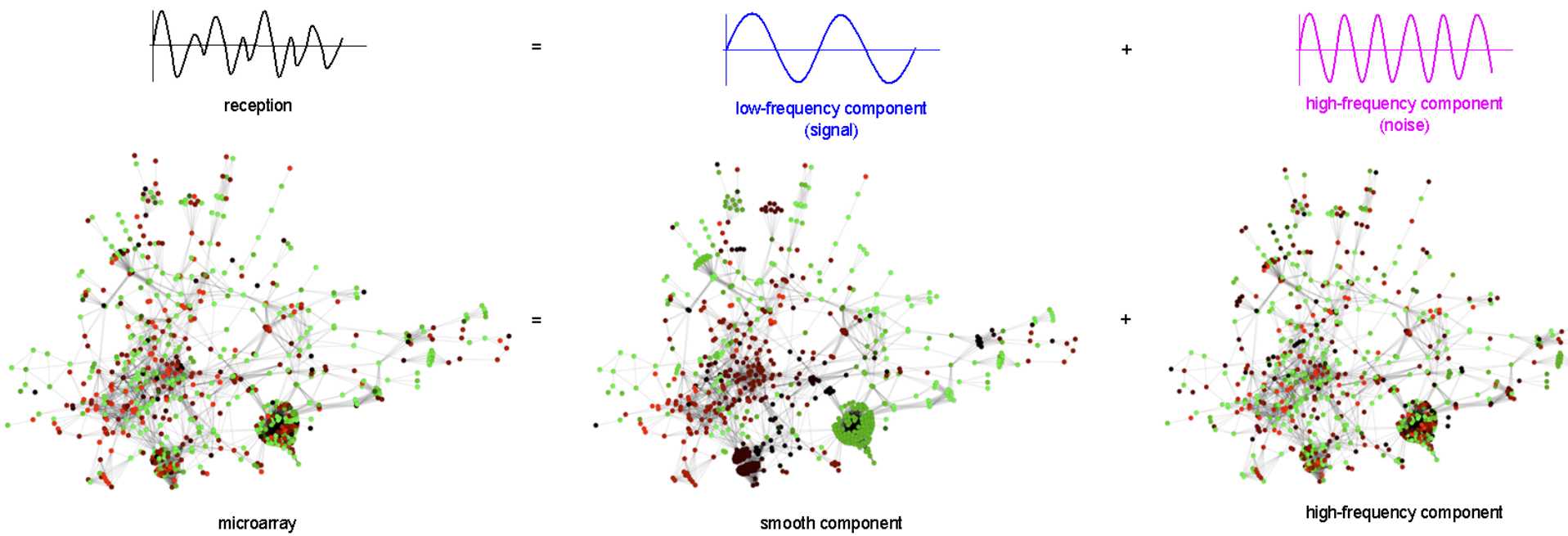
BMC Bioinformatics 2007, **8**:35 doi:10.1186/1471-2105-8-35

This article is available from: http://www.biomedcentral.com/1471-2105/8/35

Received: 30 July 2006

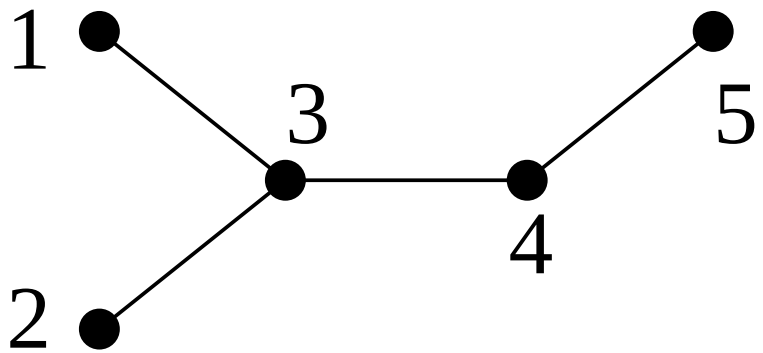
Accepted: 1 February 2007

The idea



1. Use the gene network to extract the "important information" in gene expression profiles by Fourier analysis on the graph
2. Learn a linear classifier on the smooth components

Notations

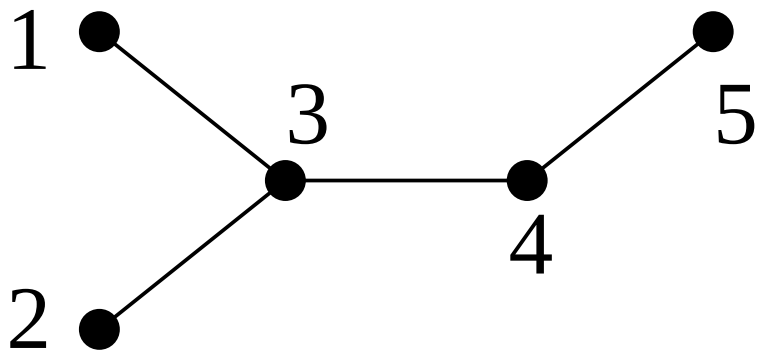


$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \qquad D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Graph Laplacian

Definition

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



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

Properties of the Laplacian

Lemma

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

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

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

- *L is a symmetric positive semi-definite matrix*
- *0 is an eigenvalue with multiplicity equal to the number of connected components.*

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

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

Proof: eigenstructure of L

- L is symmetric because A and D are symmetric.
- For any $f \in \mathbb{R}^m$, $f^\top L f \geq 0$, 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).

Fourier basis

Definition

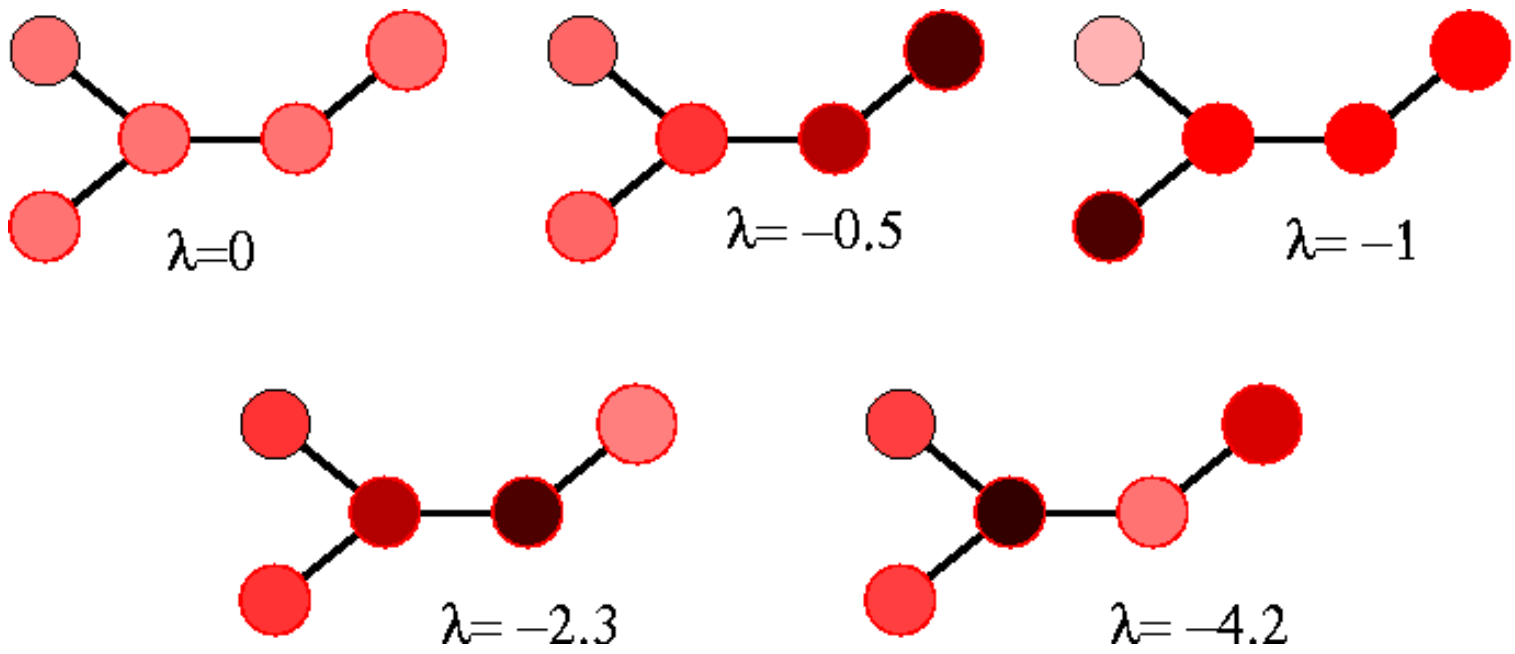
- The eigenvectors $e_1, \ldots, e_n$ of L with eigenvalues $0 = \lambda_1 \leq \ldots \leq \lambda_n$ form a basis called Fourier basis
- For any $f : V \to \mathbb{R}$, the Fourier transform of f is the vector $\hat{f} \in \mathbb{R}^n$ defined by:

$$\hat{f}_i = f^\top e_i \,, \quad i = 1, \ldots, n.$$

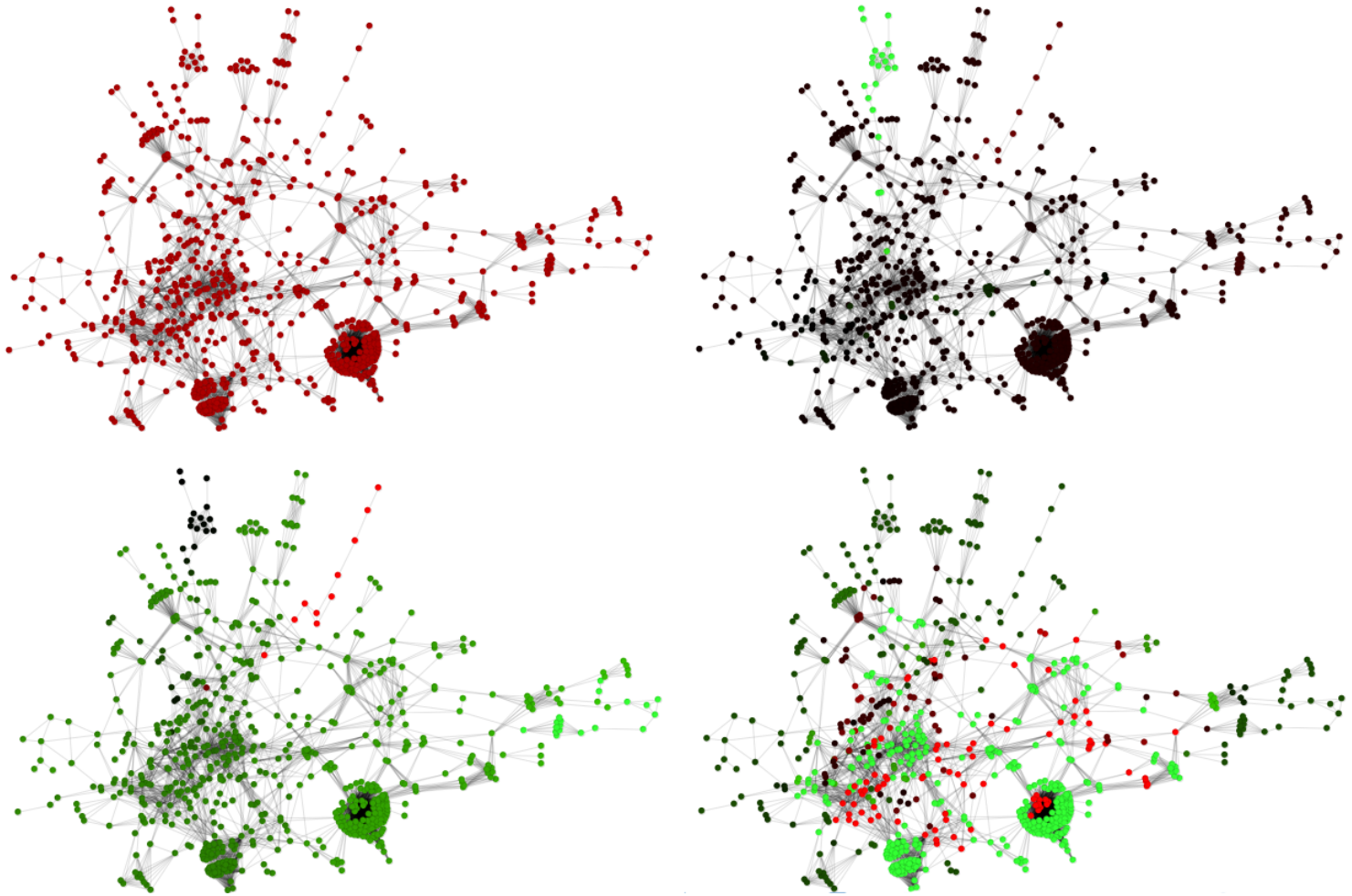
- Obviously the inverse Fourier formula holds:

$$f = \sum_{i=1}^{n} \hat{f}_i e_i \,.$$

Fourier basis



Fourier basis



Smoothing operator

Definition

- Let $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ be non-increasing.
- A smoothing operator S_ϕ transform a function $f : V \to \mathbb{R}$ into a smoothed version:

$$S_\phi(f) = \sum_{i=1}^{n} \hat{f}_i \phi(\lambda_i) e_i \,.$$

Smoothing operators

Examples

- Identity operator ($S_\phi(f) = f$):

$$\phi(\lambda) = 1\,, \quad \forall \lambda$$

- Low-pass filter:

$$\phi(\lambda) = \begin{cases} 1 & \text{if } \lambda \leq \lambda^*\,, \\ 0 & \text{otherwise.} \end{cases}$$

- Attenuation of high frequencies:

$$\phi(\lambda) = \exp(-\beta\lambda)\,.$$

Smoothing operators

Examples

- Identity operator ($S_\phi(f) = f$):

$$\phi(\lambda) = 1\,, \quad \forall \lambda$$

- Low-pass filter:

$$\phi(\lambda) = \begin{cases} 1 & \text{if } \lambda \leq \lambda^*\,, \\ 0 & \text{otherwise.} \end{cases}$$

- Attenuation of high frequencies:

$$\phi(\lambda) = \exp(-\beta\lambda)\,.$$

Smoothing operators

Examples

- Identity operator ($S_\phi(f) = f$):

$$\phi(\lambda) = 1\,, \quad \forall \lambda$$

- Low-pass filter:

$$\phi(\lambda) = \begin{cases} 1 & \text{if } \lambda \leq \lambda^*\,, \\ 0 & \text{otherwise.} \end{cases}$$

- Attenuation of high frequencies:

$$\phi(\lambda) = \exp(-\beta\lambda)\,.$$

Supervised classification and regression

Working with smoothed profiles

- Classical methods for linear classification and regression with a ridge penalty solve:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} l\left(\beta^\top f_i, y_i\right) + \lambda \beta^\top \beta \,.$$

- Applying these algorithms on the smooth profiles means solving:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} l\left(\beta^\top S_\phi(f_i), y_i\right) + \lambda \beta^\top \beta \,.$$

Smooth solution

Lemma

This is equivalent to:

$$\min_{v \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} l\left(v^\top f_i, y_i\right) + \lambda \sum_{i=1}^{p} \frac{\hat{v}_i^2}{\phi(\lambda_i)} \,,$$

hence the linear classifier v is smooth.

Proof

- Let $v = \sum_{i=1}^n \phi(\lambda_i) e_i e_i^\top \beta$, then

$$\beta^\top S_\phi(f_i) = \beta^\top \sum_{i=1}^{n} \hat{f}_i \phi(\lambda_i) e_i = f^\top v \,.$$

- Then $\hat{v}_i = \phi(\lambda_i)\hat{\beta}_i$ and $\beta^\top \beta = \sum_{i=1}^n \frac{\hat{v}_i^2}{\phi(\lambda_i)^2}$.

Smooth solution

Lemma

This is equivalent to:

$$\min_{v \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} l\left(v^\top f_i, y_i\right) + \lambda \sum_{i=1}^{p} \frac{\hat{v}_i^2}{\phi(\lambda_i)} \,,$$

hence the linear classifier v is smooth.

Proof

- Let $v = \sum_{i=1}^n \phi(\lambda_i) e_i e_i^\top \beta$, then

$$\beta^\top S_\phi(f_i) = \beta^\top \sum_{i=1}^{n} \hat{f}_i \phi(\lambda_i) e_i = f^\top v \,.$$

- Then $\hat{v}_i = \phi(\lambda_i)\hat{\beta}_i$ and $\beta^\top \beta = \sum_{i=1}^n \frac{\hat{v}_i^2}{\phi(\lambda_i)^2}$.

Smoothing kernel

Kernel methods (SVM, kernel ridge regression..) only need the inner product between smooth profiles:

$$
\begin{aligned}
K(f,g) &= S_\phi(f)^\top S_\phi(g) \\
&= \sum_{i=1}^{n} \hat{f}_i \hat{g}_i \phi(\lambda_i)^2 \\
&= f^\top \left(\sum_{i=1}^{n} \phi(\lambda_i)^2 e_i e_i^\top \right) g \\
&= f^\top K_\phi g \,,
\end{aligned}
\tag{4}
$$

with

$$
K_\phi = \sum_{i=1}^{n} \phi(\lambda_i)^2 e_i e_i^\top \,.
$$

Examples

- For $\phi(\lambda) = \exp(-t\lambda)$, we recover the diffusion kernel:

$$K_\phi = \exp_M(-2tL)\,.$$

- For $\phi(\lambda) = 1/\sqrt{1+\lambda}$, we obtain

$$K_\phi = (L + I)^{-1}\,,$$

and the penalization is:

$$\sum_{i=1}^{n} \frac{\hat{v}_i^2}{\phi(\lambda_i)} = v^\top (L + I)\, v = \| v \|_2^2 + \sum_{i \sim j} (v_i - v_j)^2\,.$$

Examples

- For $\phi(\lambda) = \exp(-t\lambda)$, we recover the diffusion kernel:

$$K_\phi = \exp_M(-2tL)\,.$$

- For $\phi(\lambda) = 1/\sqrt{1+\lambda}$, we obtain

$$K_\phi = (L + I)^{-1}\ ,$$

and the penalization is:

$$\sum_{i=1}^{n} \frac{\hat{v}_i^2}{\phi(\lambda_i)} = v^\top (L + I)\, v = \| v \|_2^2 + \sum_{i \sim j} (v_i - v_j)^2\,.$$

Outline

Data

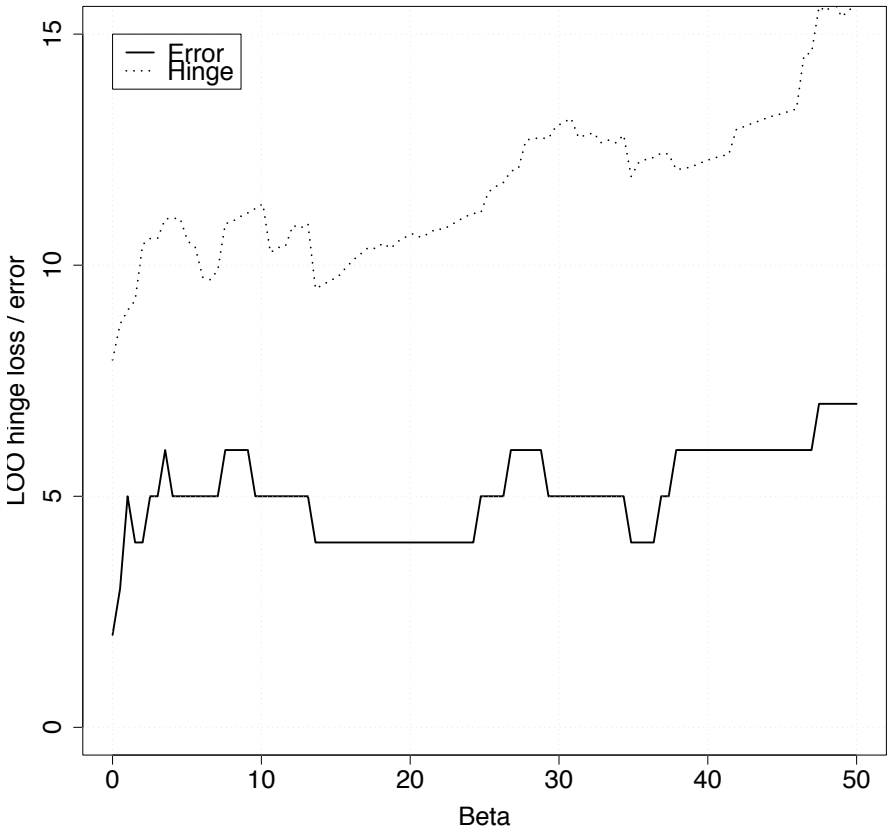
Expression

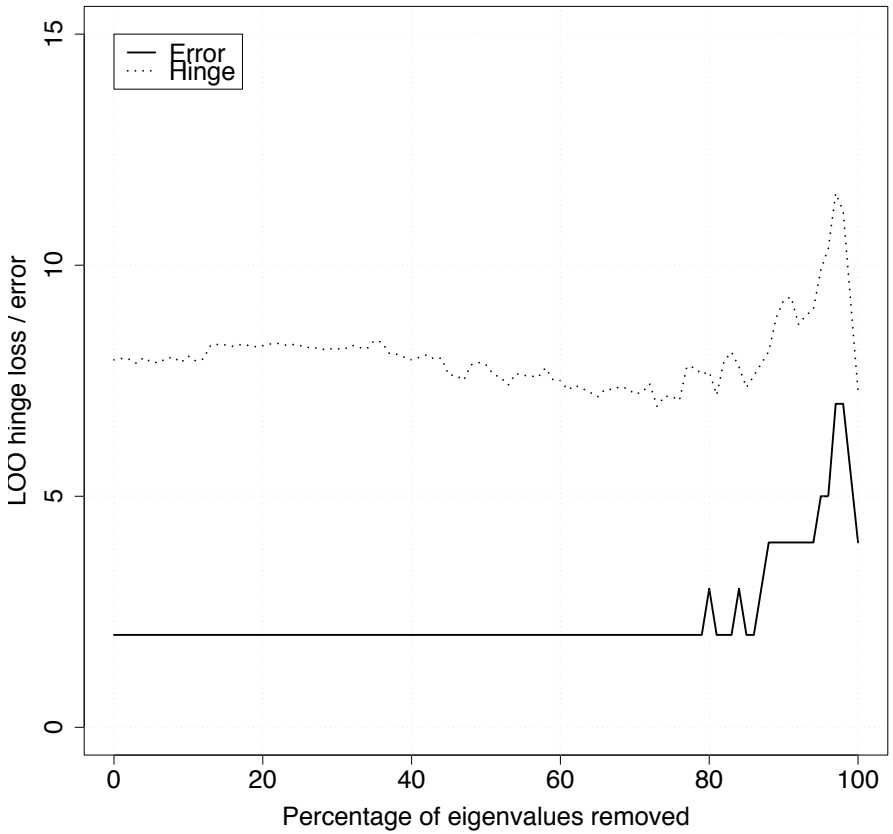
- Study the effect of low irradiation doses on the yeast
- 12 non irradiated vs 6 irradiated
- Which pathways are involved in the response at the transcriptomic level?

Graph

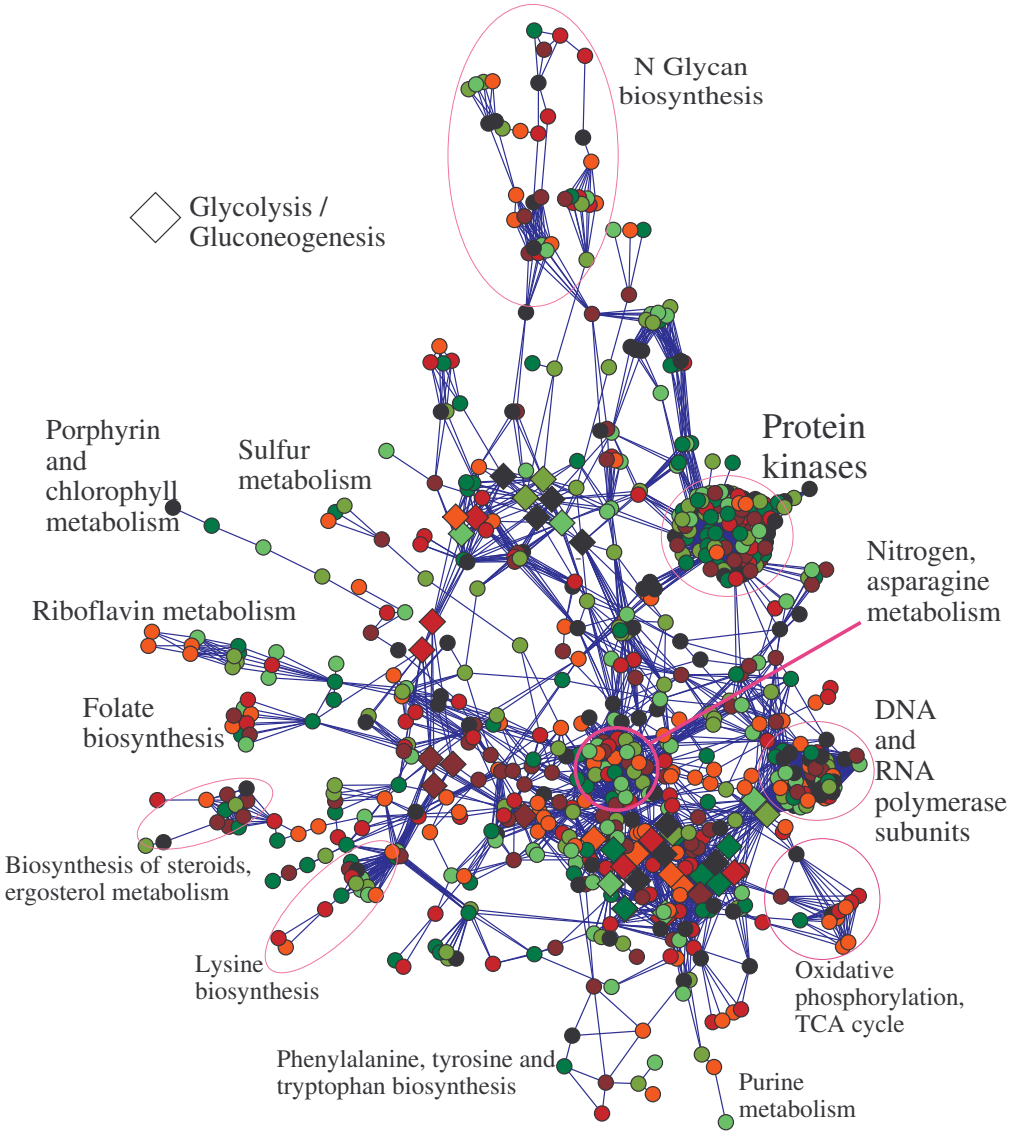
- KEGG database of metabolic pathways
- Two genes are connected is they code for enzymes that catalyze successive reactions in a pathway (metabolic gene network).
- 737 genes, 4694 vertices.

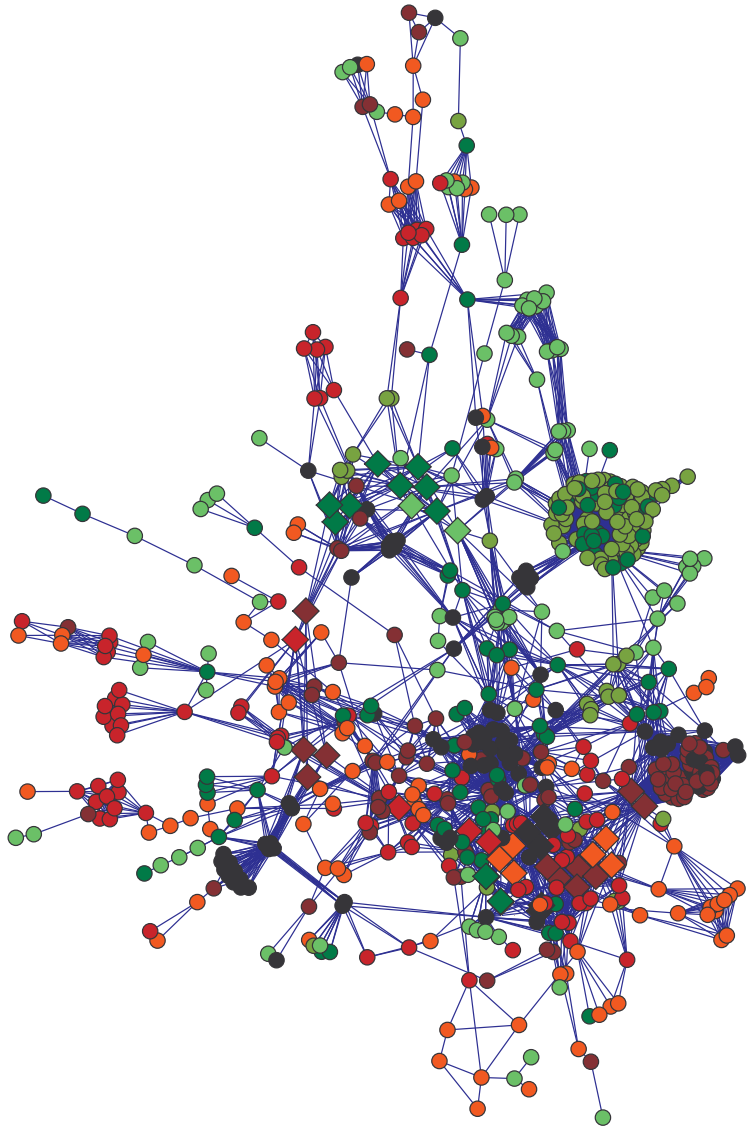
Classification performance



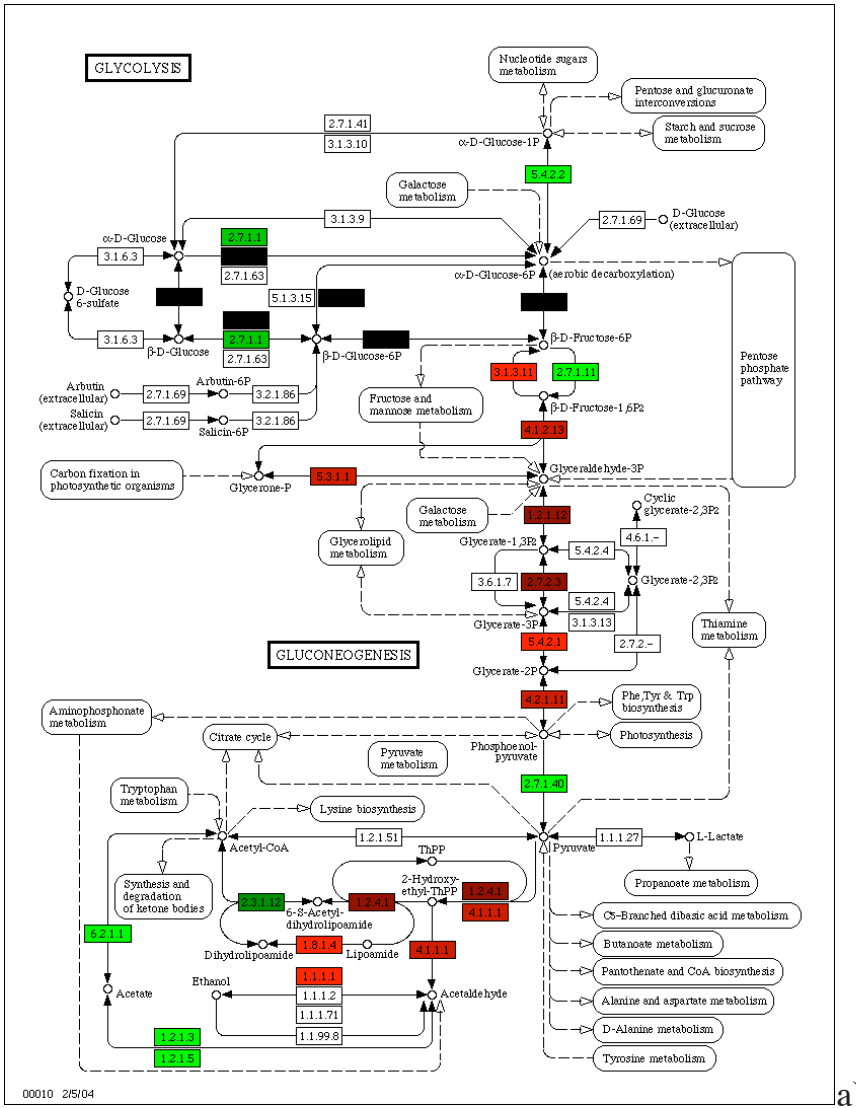


Classifier

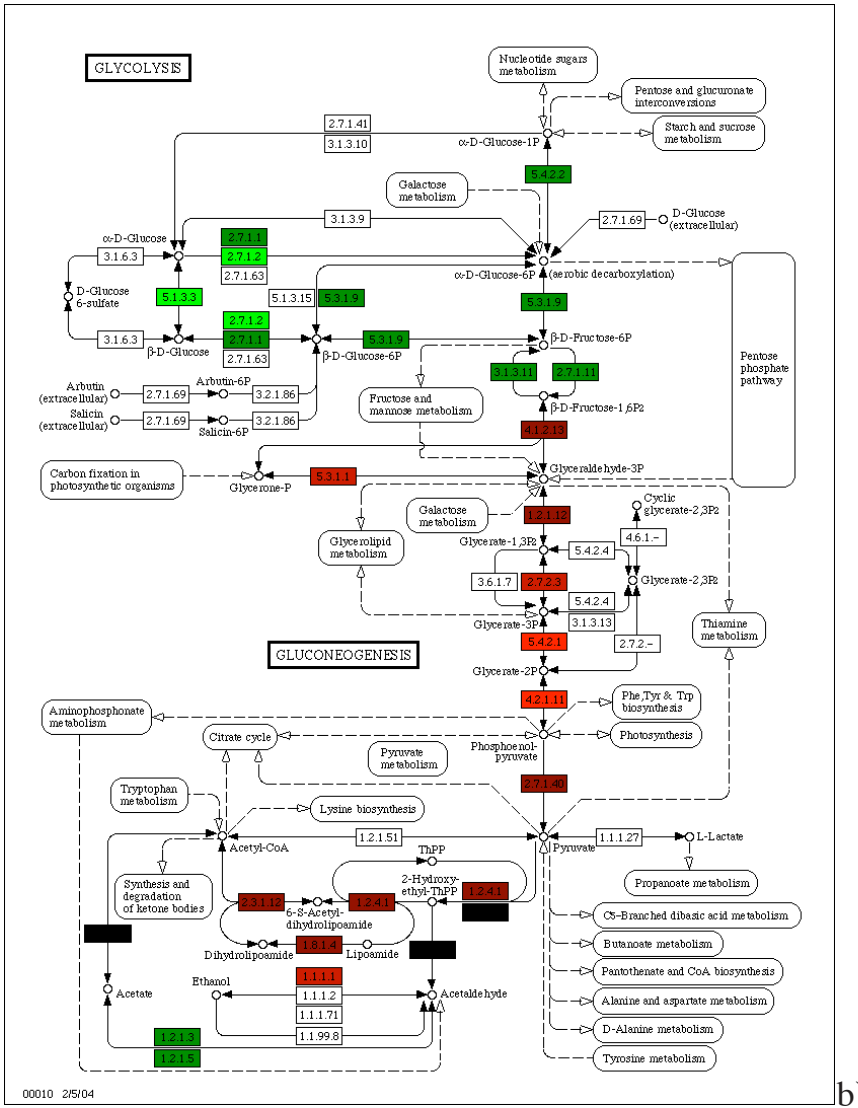

N Glycan biosynthesis
Glycolysis / Gluconeogenesis
Porphyrin and chlorophyll metabolism
Sulfur metabolism
Protein kinases
Nitrogen, asparagine metabolism
Riboflavin metabolism
Folate biosynthesis
DNA and RNA polymerase subunits
Biosynthesis of steroids, ergosterol metabolism
Lysine biosynthesis
Oxidative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



Classifier



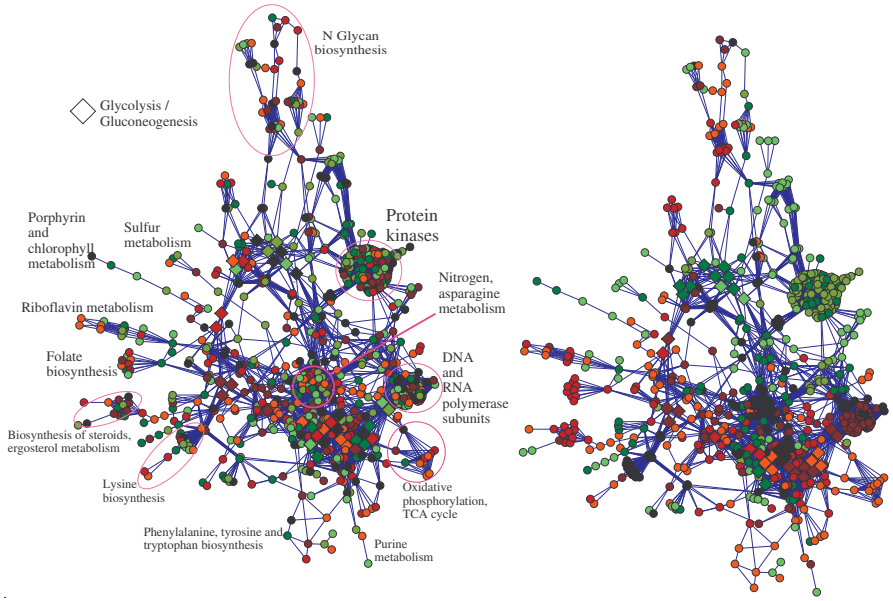
a)



b)

Summary

- Given a gene network, spectral graph analysis (Fourier analysis) is helpful to analyze signals over the network, e.g., gene expression data
- We can smooth profiles with frequency filters or attenuation
- Combined with a SVM through spectral graph kernels, we can detect discriminant pathways or protein complexes.



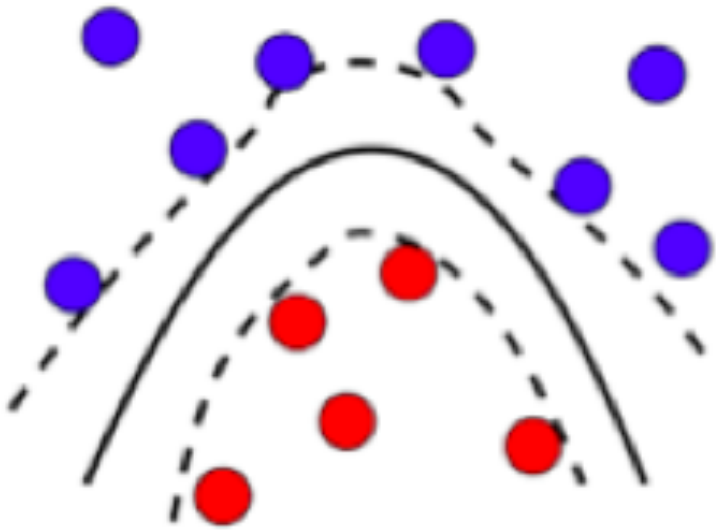
Outline

Machine learning in computational and systems biology

- Biology faces a flood of data following the development of high-throughput technologies (sequencing, DNA chips, ...)
- Many problems can be formalized in the framework of machine learning, e.g.:
  - Protein annotation
  - Drug discovery, virtual screening
  - Gene network inference
- These data have often complex structures (strings, graphs, high-dimensional vectors) and often require dedicated algorithms.

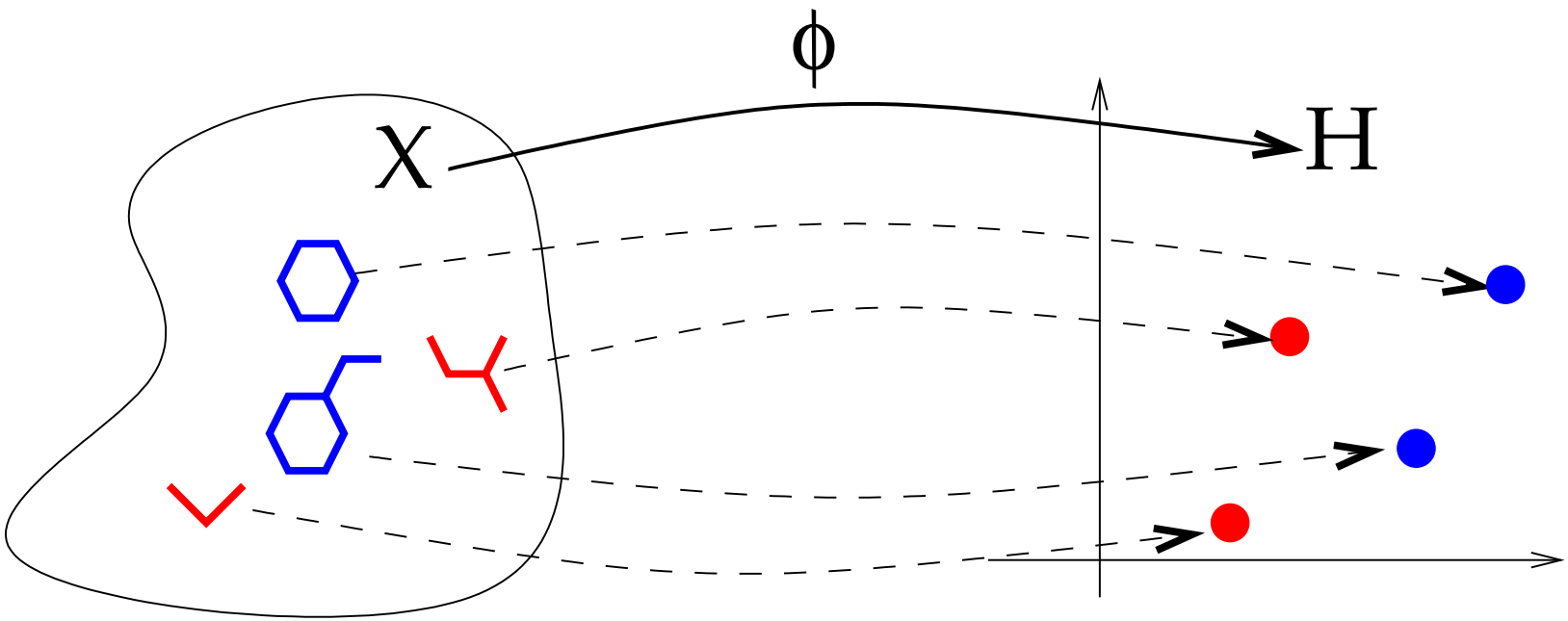
Support vector machines (SVM)

- A general-purpose algorithm for pattern recognition
- Based on the principle of large margin *("séparateur à vaste marge")*
- Linear or nonlinear with the kernel trick
- Control of the regularization / data fitting trade-off with the C parameter
- State-of-the-art performance on many applications



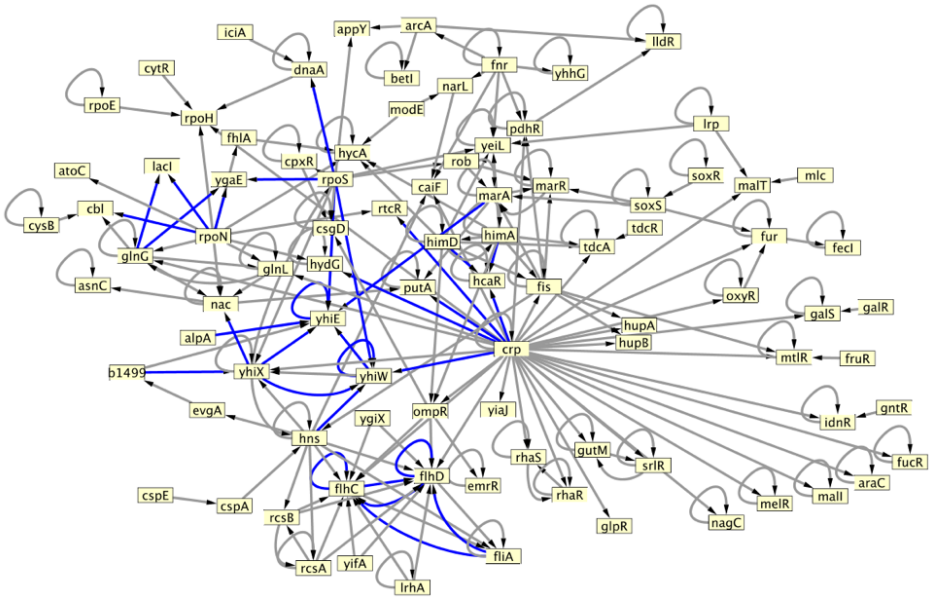
Kernels

- A central ingredient of SVM
- Allows nonlinearity
- Allows to work implicitly in a high-dimensional feature space
- Allows to work with structured data (e.g., graphs)



Gene network inference

- *Ab initio* reconstruction of regulatory network can be formulated as feature selection, and solved, e.g., by the Lasso or random forests
- Supervised reconstruction is more powerful when edges (e.g., regulations) are already known
- PU learning is more powerful than one-class learning in this setting, and can be solved by SVM
- Predicting edges requires learning over pairs with specific kernels in the case of SVM



Using gene networks

- Gene networks can be used as prior knowledge to analyze gene expression data
- Spectral graph analysis and graph kernels are useful tools
- It allows to capture pathways or protein complexes instead of individual genes