Inference of biological networks: from *de novo* to supervised approaches

Jean-Philippe Vert
Jean-Philippe.Vert@mines.org

Mines ParisTech / Institut Curie / Inserm

UC London, Jan 7, 2011.

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Gene expression

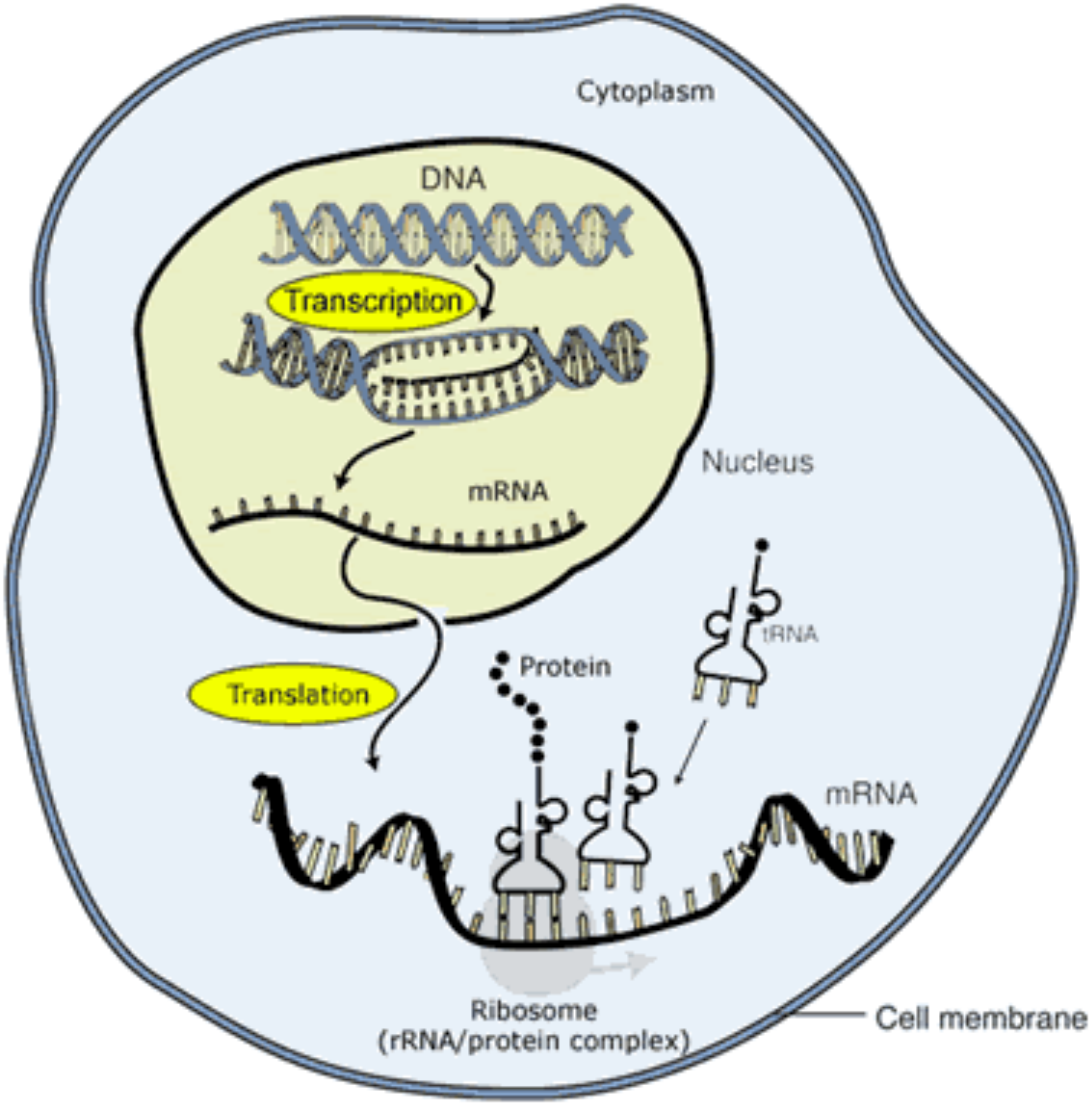
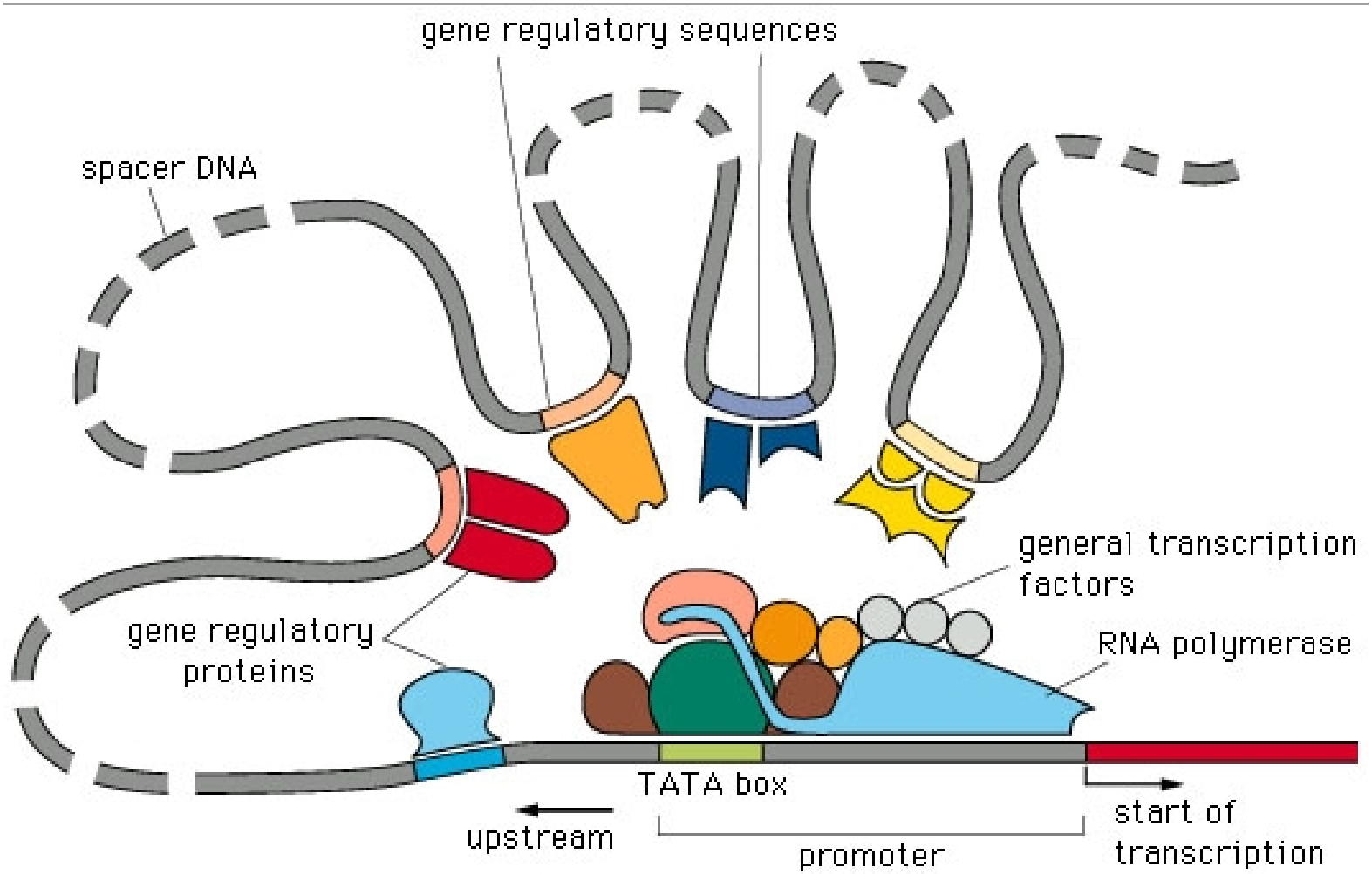
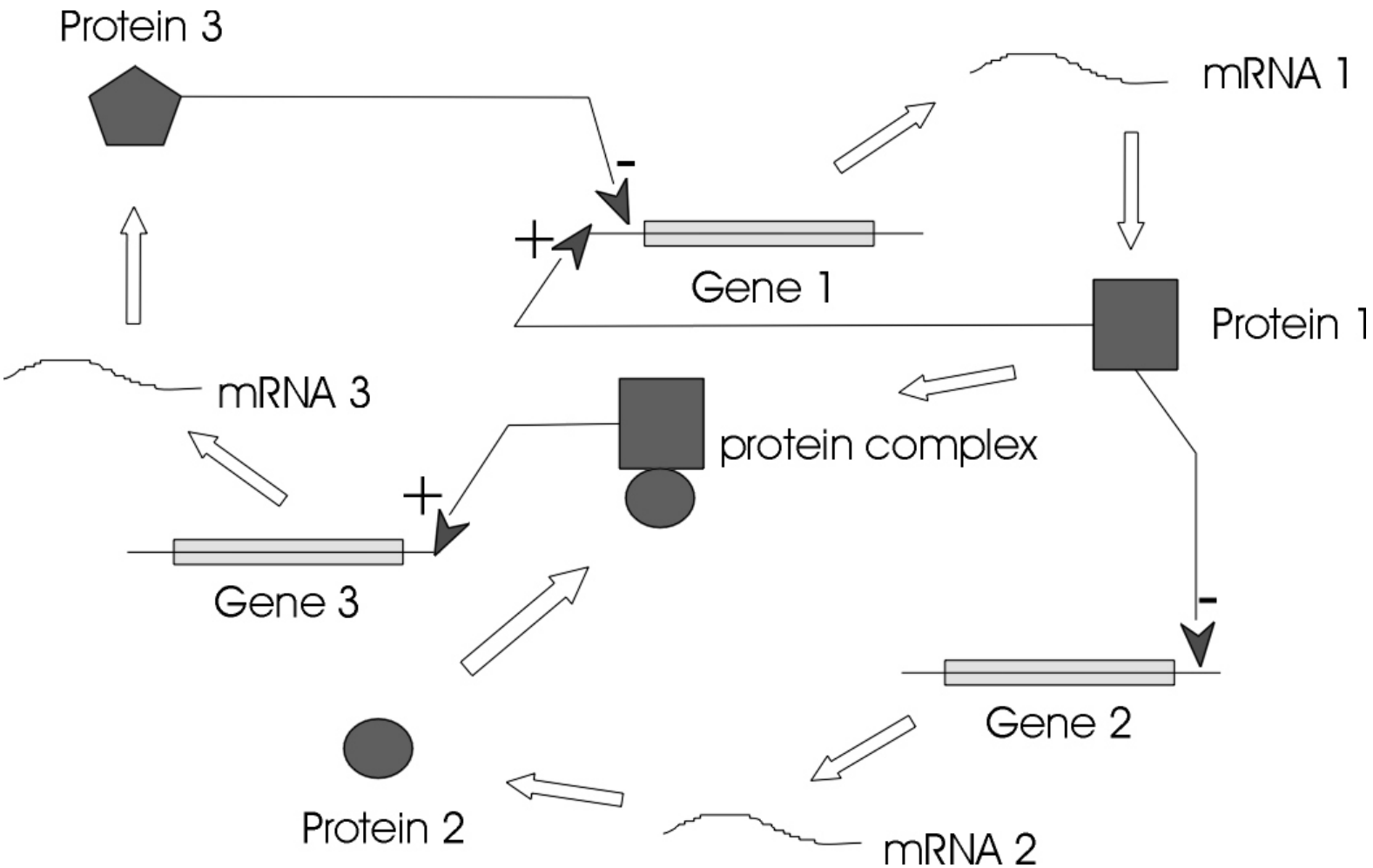


Image adapted from: National Human Genome Research Institute.

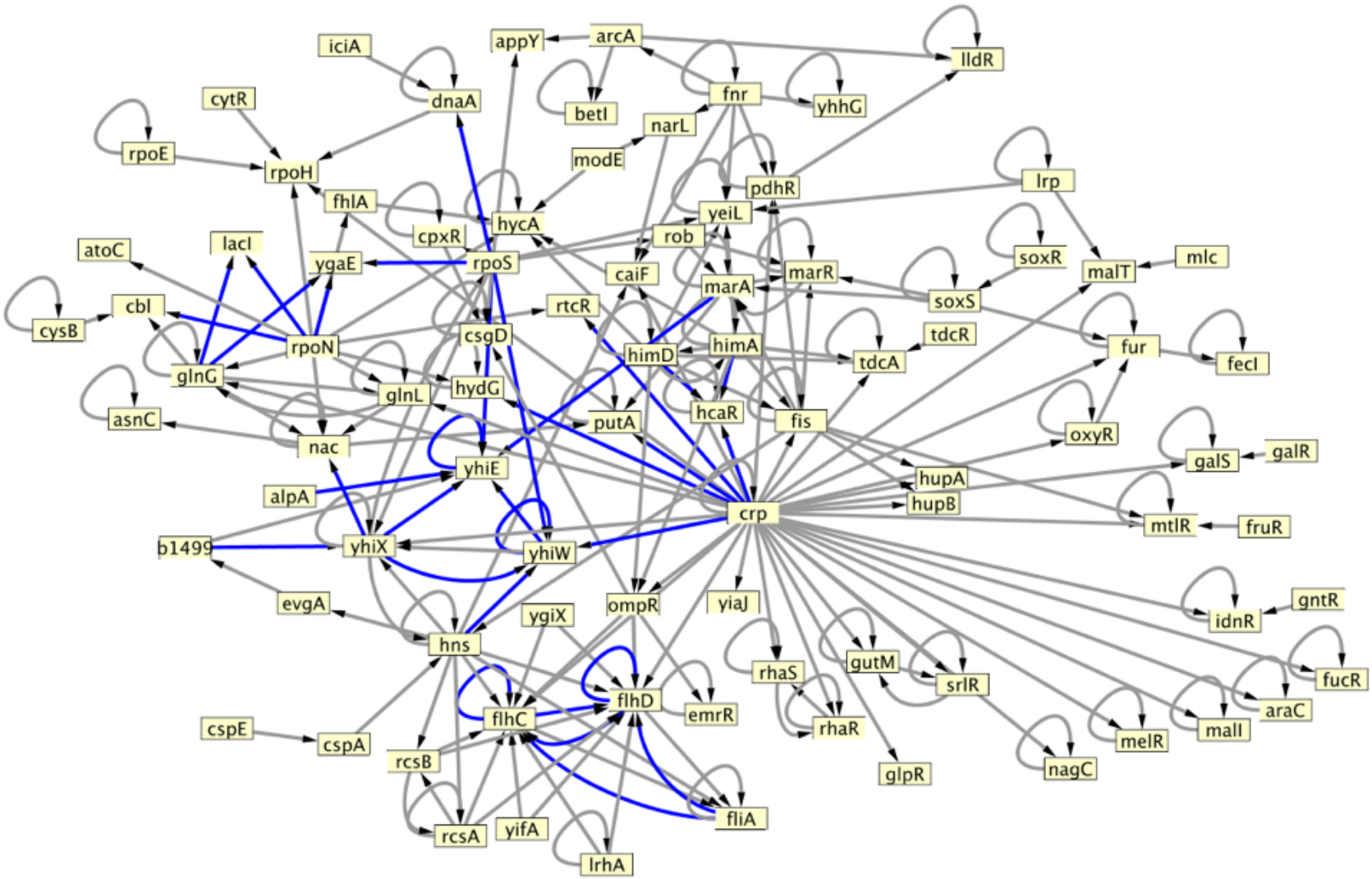
Gene expression regulation



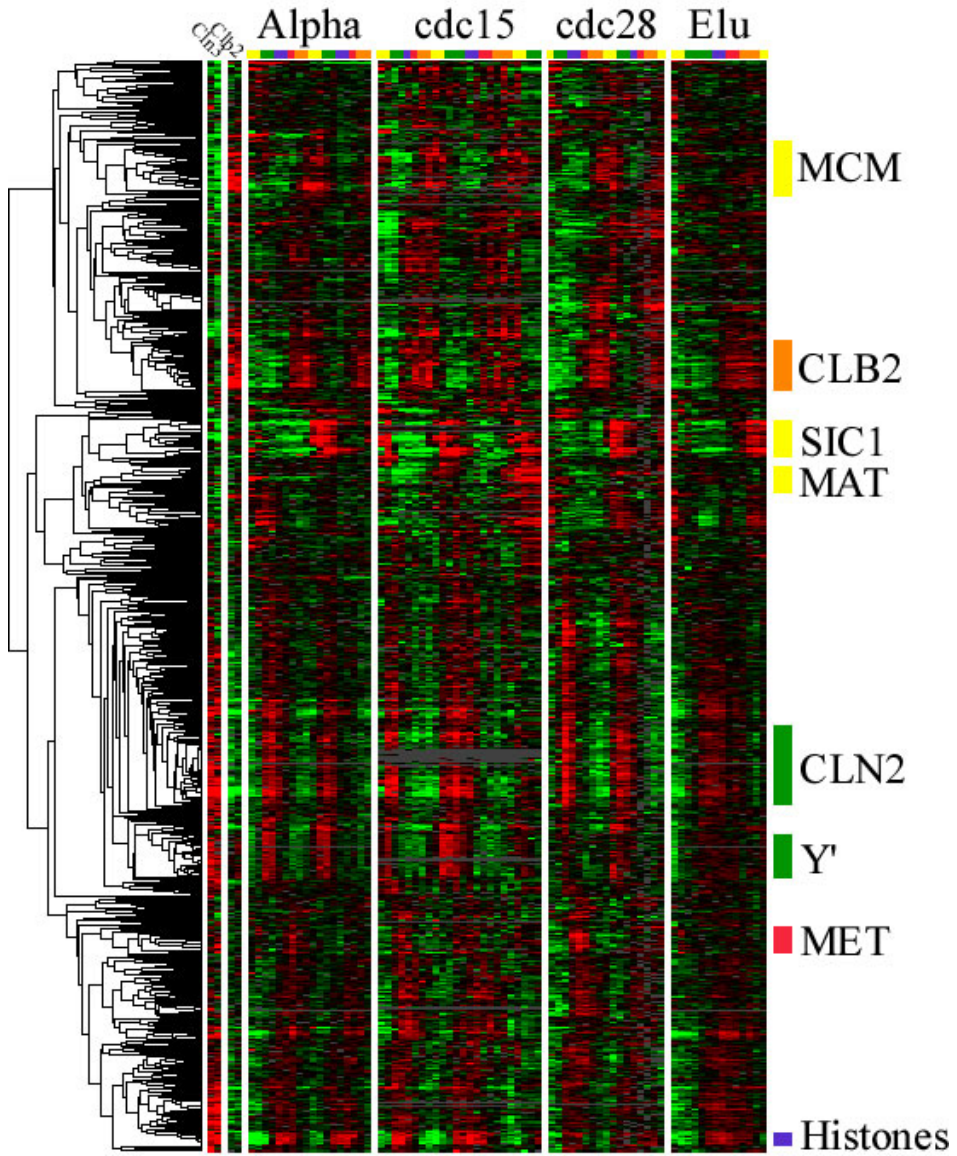
Gene regulatory network



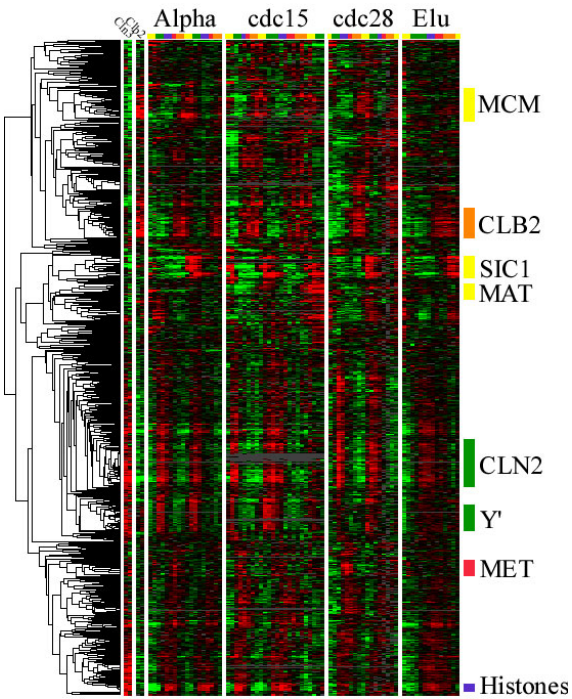
Gene regulatory network of E. coli

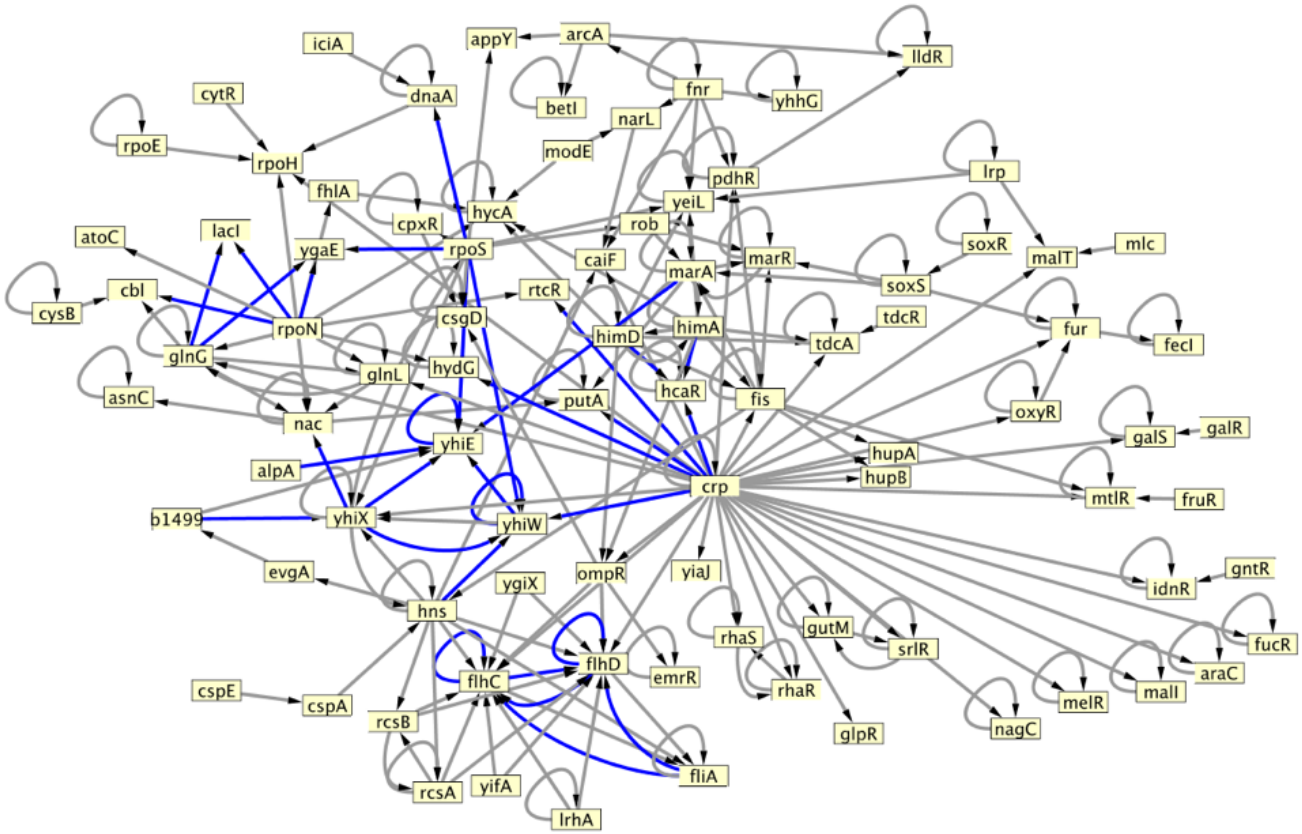


Gene expression data

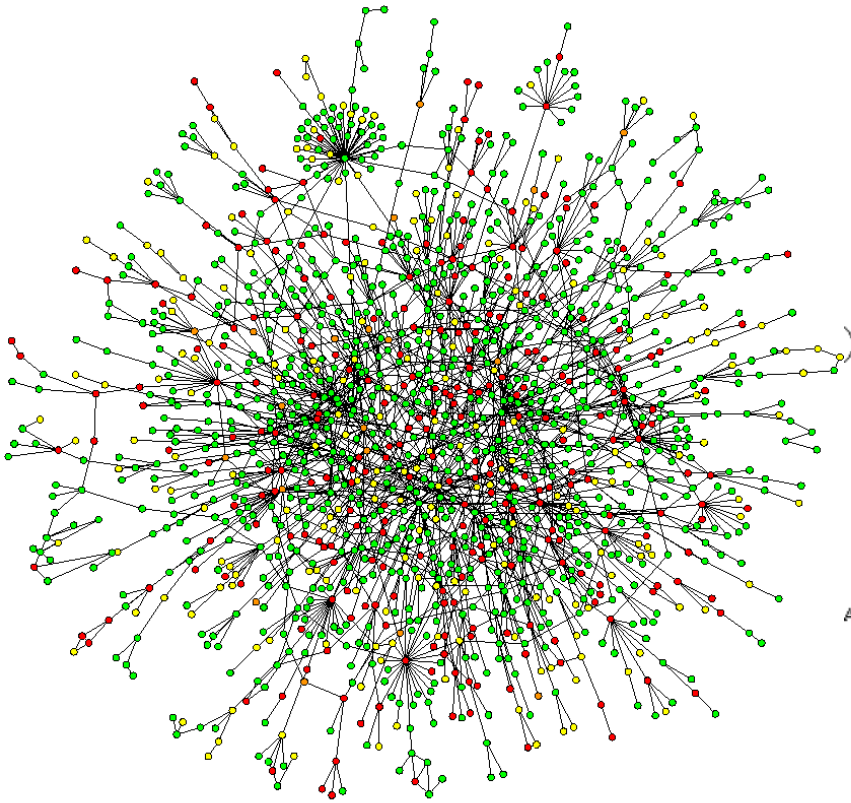


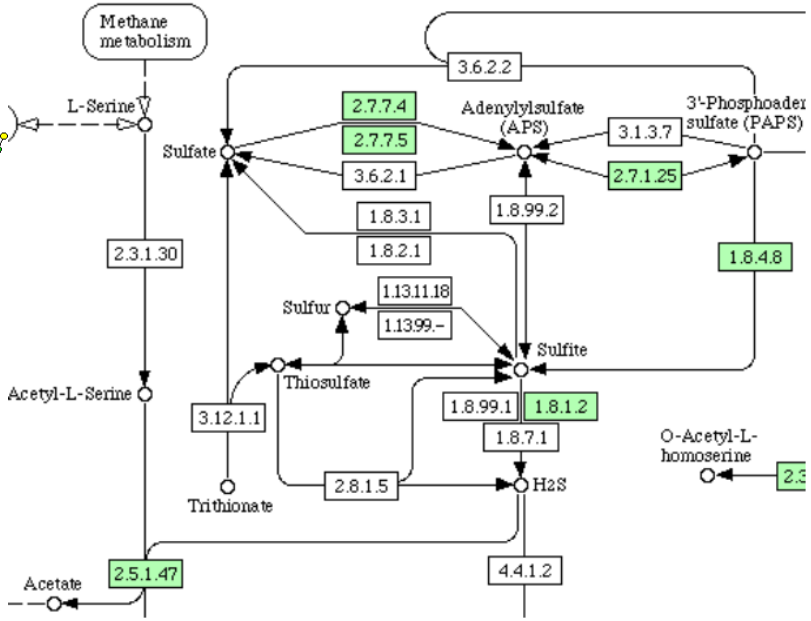
Reconstruction of gene regulatory network from expression data



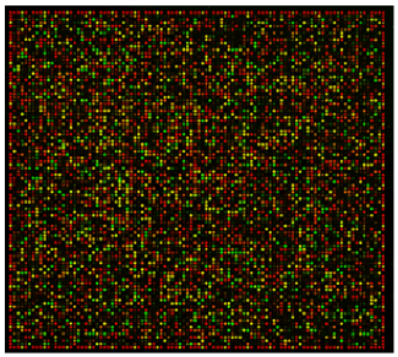


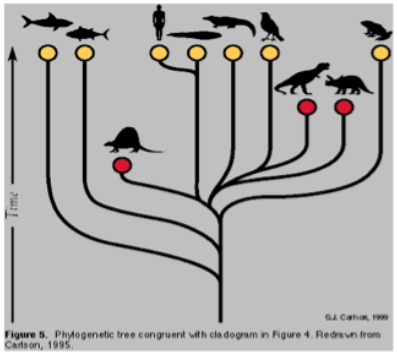
More networks...

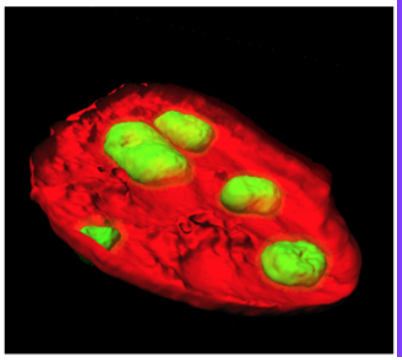


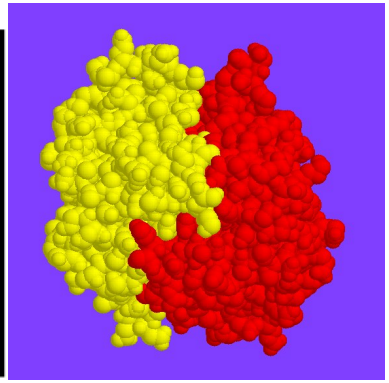


More data..



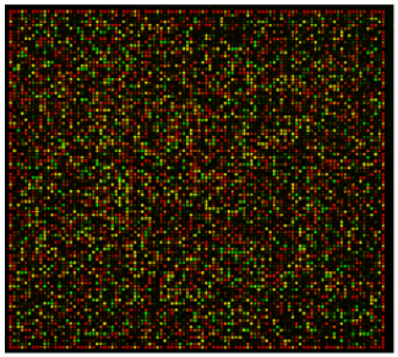


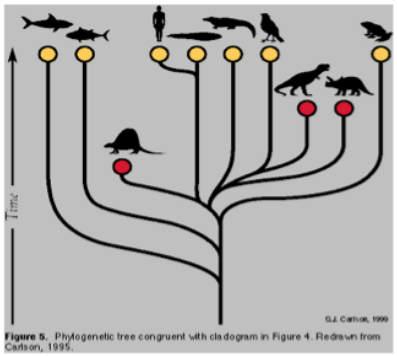


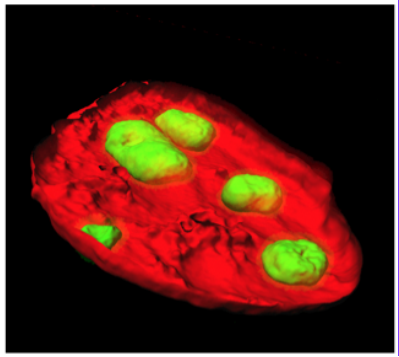


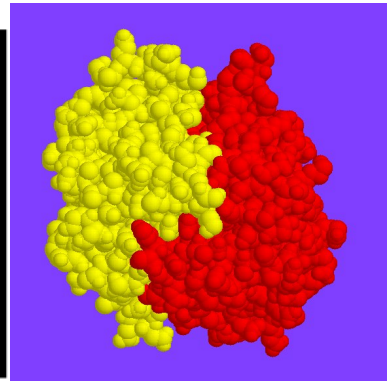
- Gene expression measurements
- Phylogenetic profiles
- Location of proteins/enzymes in the cell
- Structures...

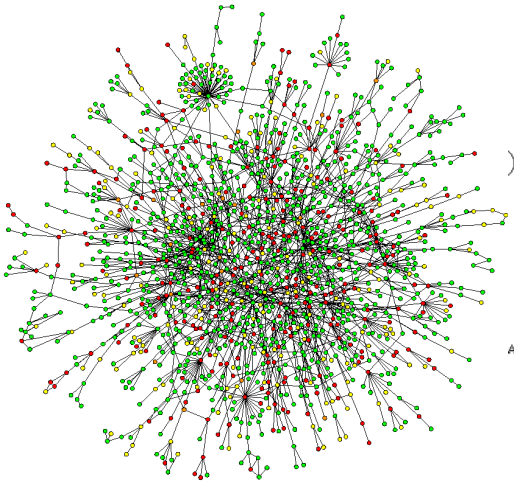
General gene network inference problem

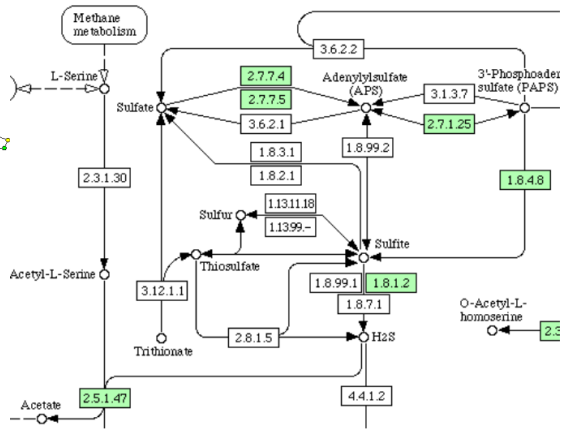


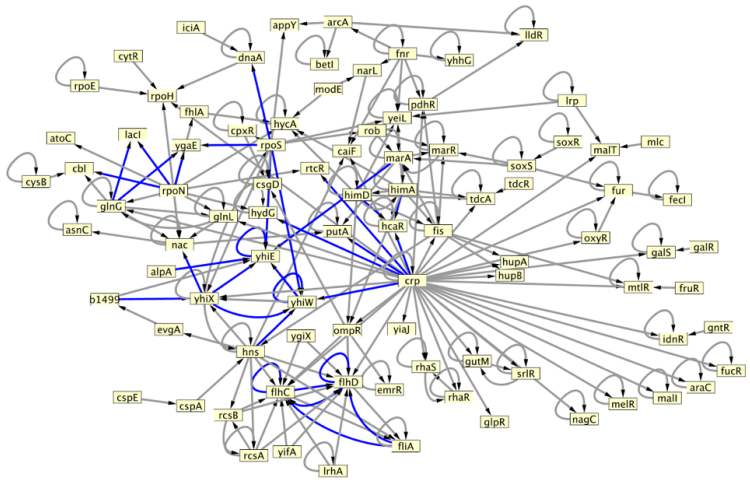










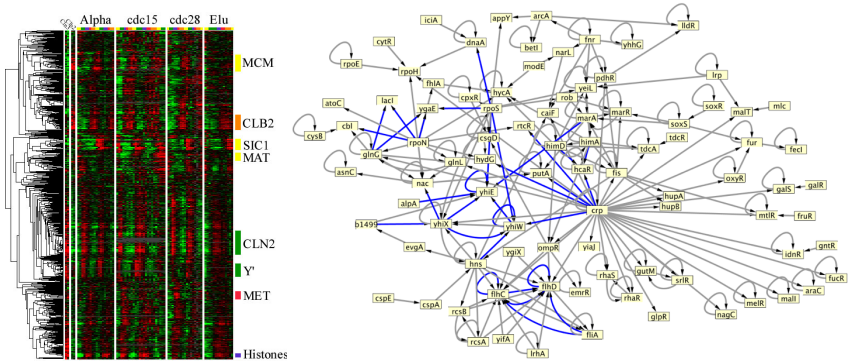


Outline

De novo inference

The problem

Given data about the genes (eg, expression), infer the edges (eg, regulations).



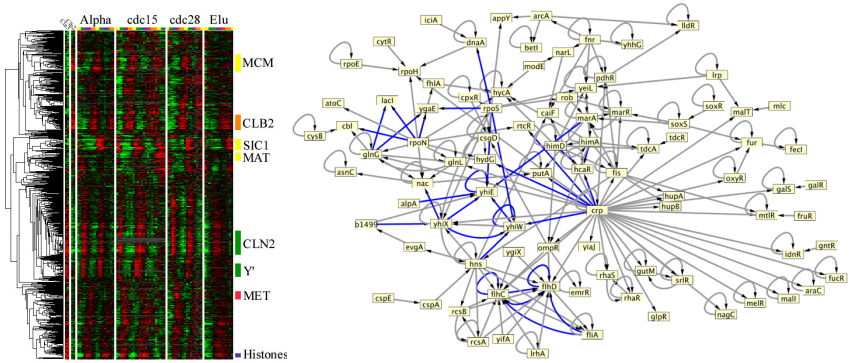
How?

- Interactions are between "similar" genes?
- Interactions are between "dependent" genes?
- Interactions are between "predictive" genes?

De novo inference

The problem

Given data about the genes (eg, expression), infer the edges (eg, regulations).



How?

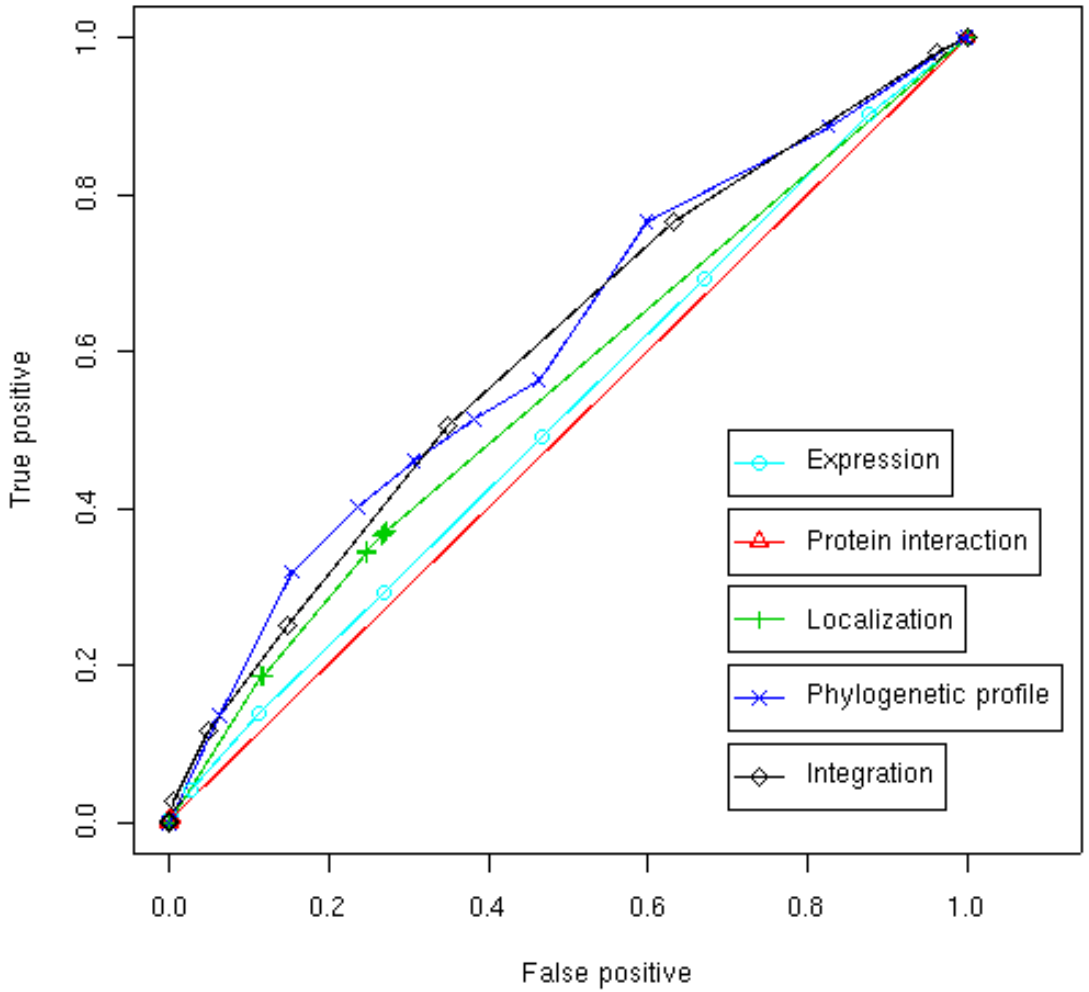
- Interactions are between "similar" genes?
- Interactions are between "dependent" genes?
- Interactions are between "predictive" genes?

Predict interactions between "similar" genes

- In most networks, connected genes are significantly more "similar" than non-connected ones
- Inference: connect genes whose similarity (eg, Euclidean distance between profiles) is above a threshold

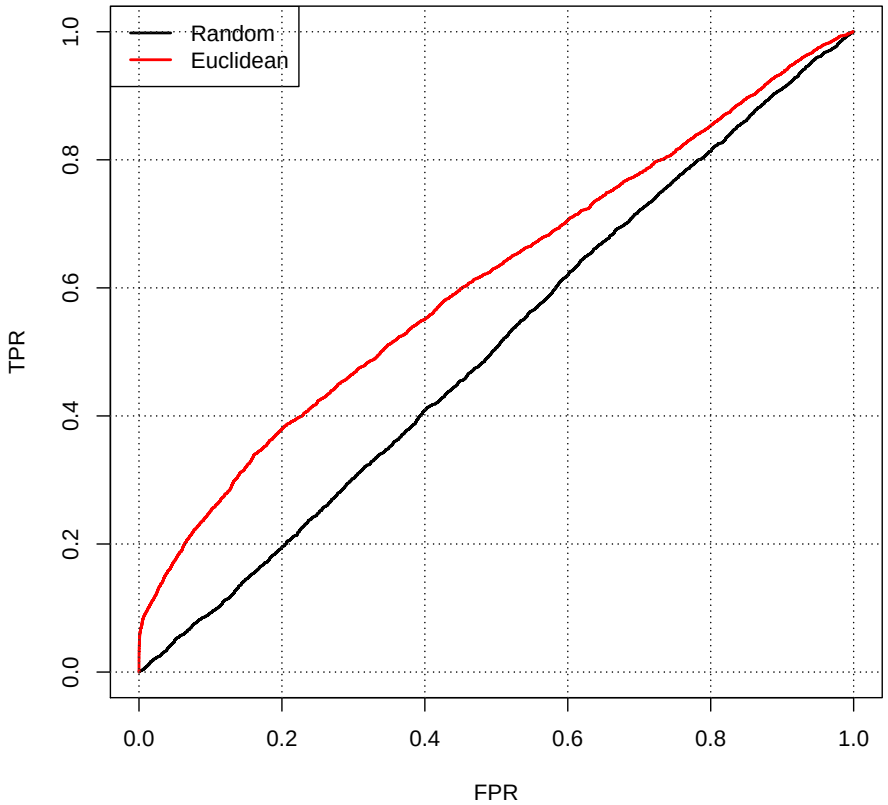
Example: yeast metabolic network

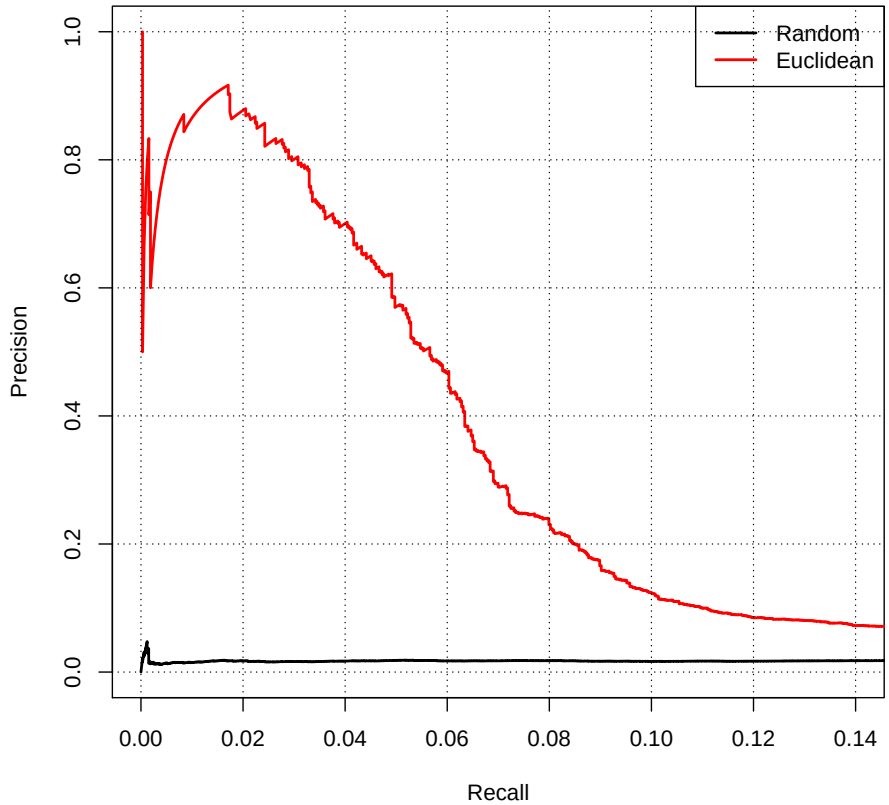
- 769 proteins, 3702 metabolic edges
- Inference: rank by decreasing similariy of expression, interactions, localization, phylogenetic profiles



Example: E coli regulatory network

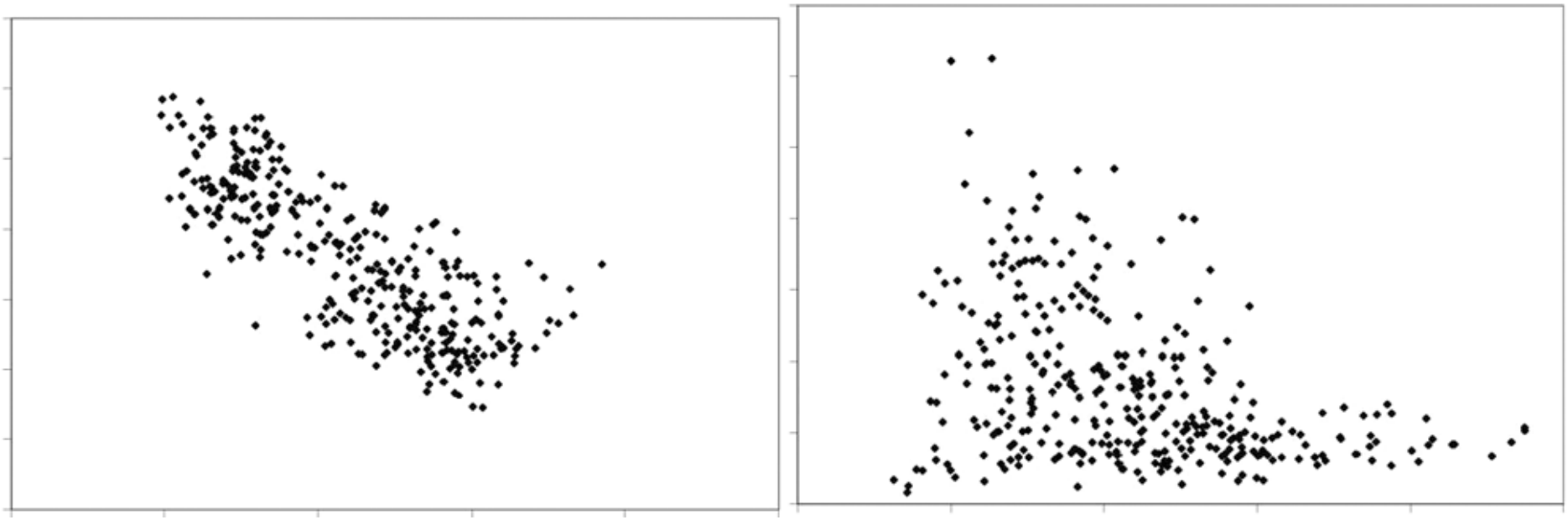
- 154 TF targeting 1164 genes through 3293 regulations
- Inference: rank by decreasing Euclidean distance between expression profiles





Predict regulations between "dependent" genes

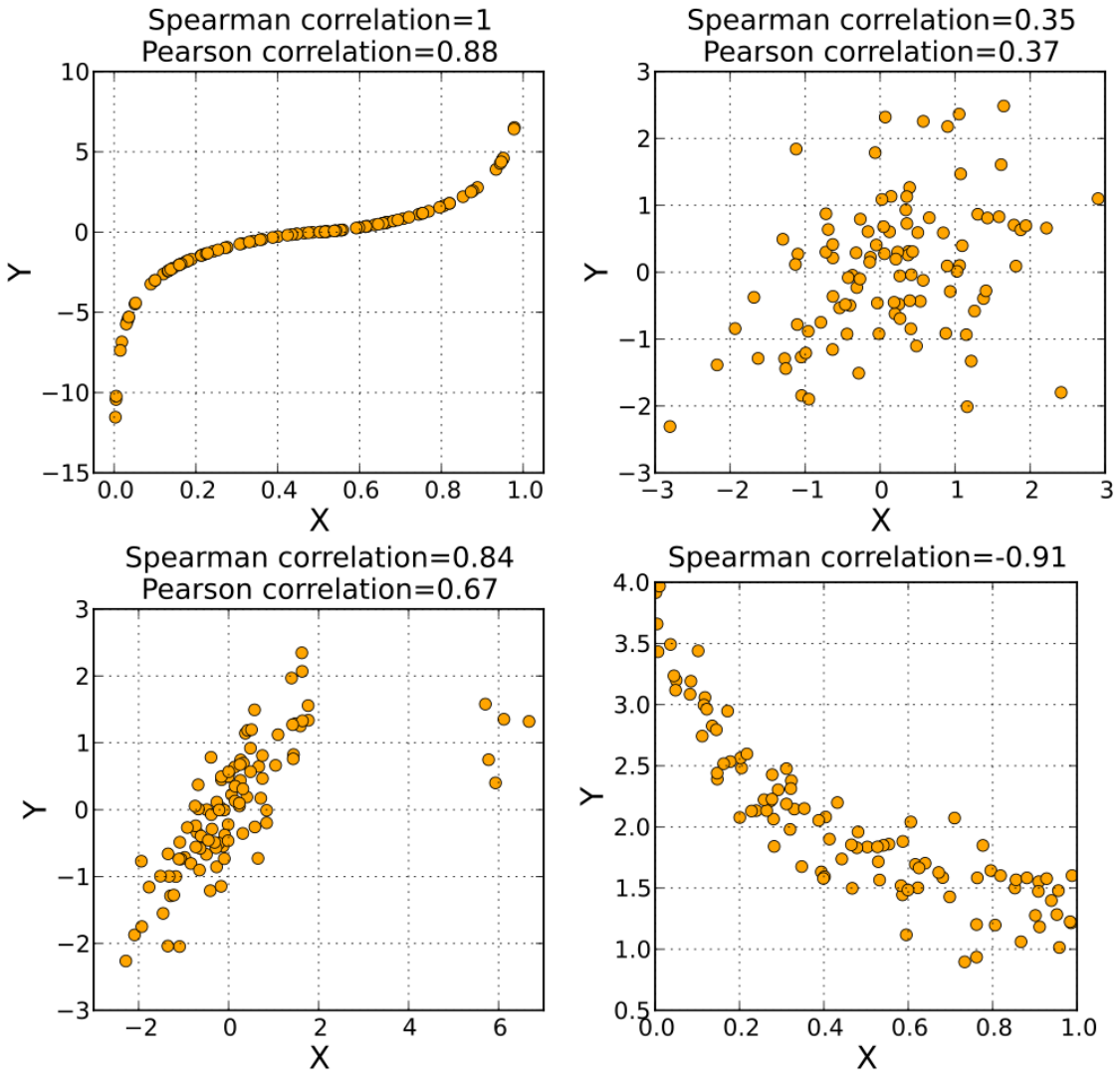
Sometimes the expression of a TF and its target are not similar, but correlated or dependent



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

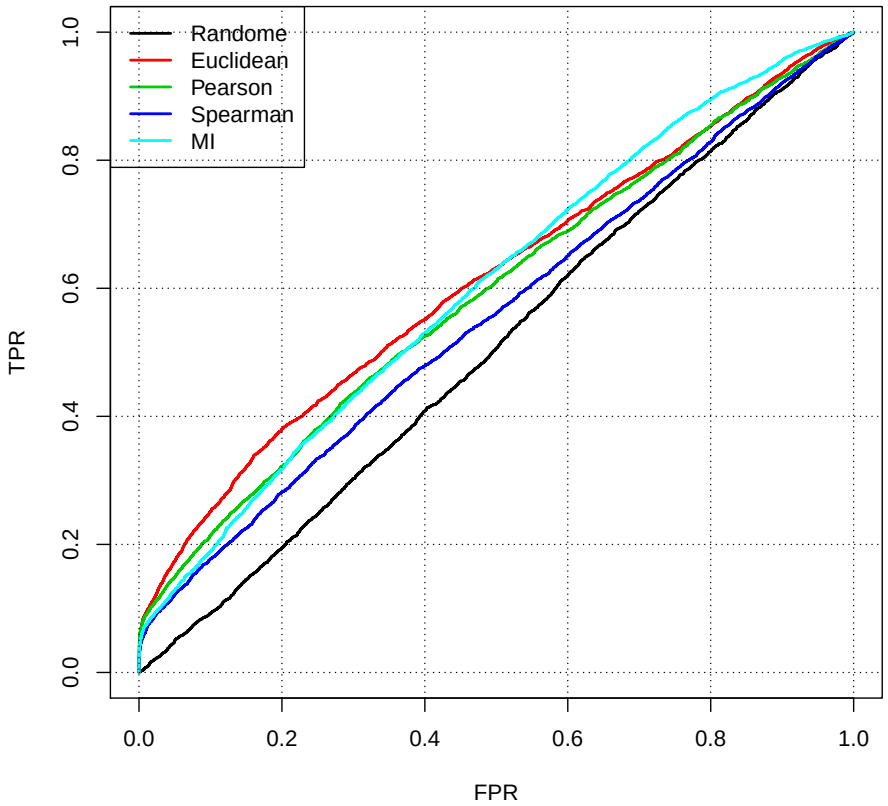
Measuring dependency

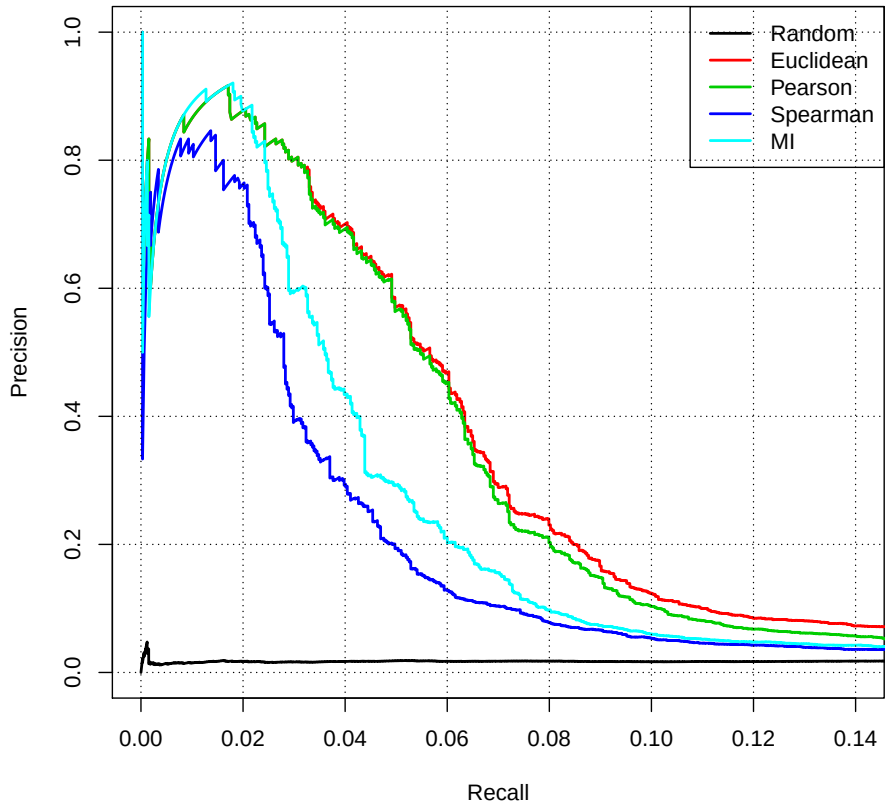
Pearson/Spearman correlation, mutual information (ARACNE, CLR...)



Validation

Application: E coli regulatory network : 154 TF targeting 1164 genes through 3293 regulations





Predict regulations between "predictive" genes

- 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 transcription factors whose expression is sufficient to explain the expression of X across different experiments.

Predicting regulation by sparse regression

- Treat each target in turn
- Let Y the expression of a target, and $X_1, \ldots, X_p$ the expression of all TFs. We look for a model

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

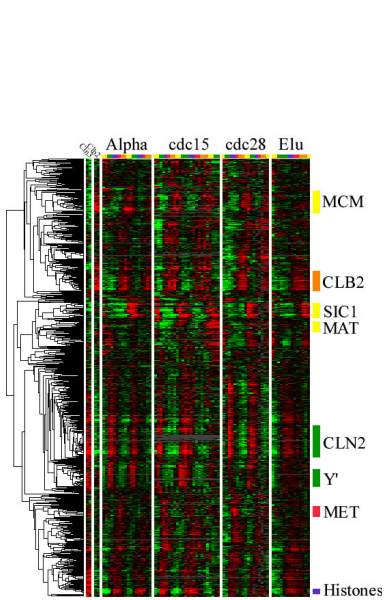
  where β is sparse, i.e., only a few β_i are non-zero
- Examples:
  - GENIE: feature selection by random forest (Huynh-Thu et al., 2010)
  - Feature selection by Lasso + stability selection (Haury et al., 2011)
- Both methods were ranked 1st and 2nd (out of 28) at the DREAM5 in silico network inference challenge

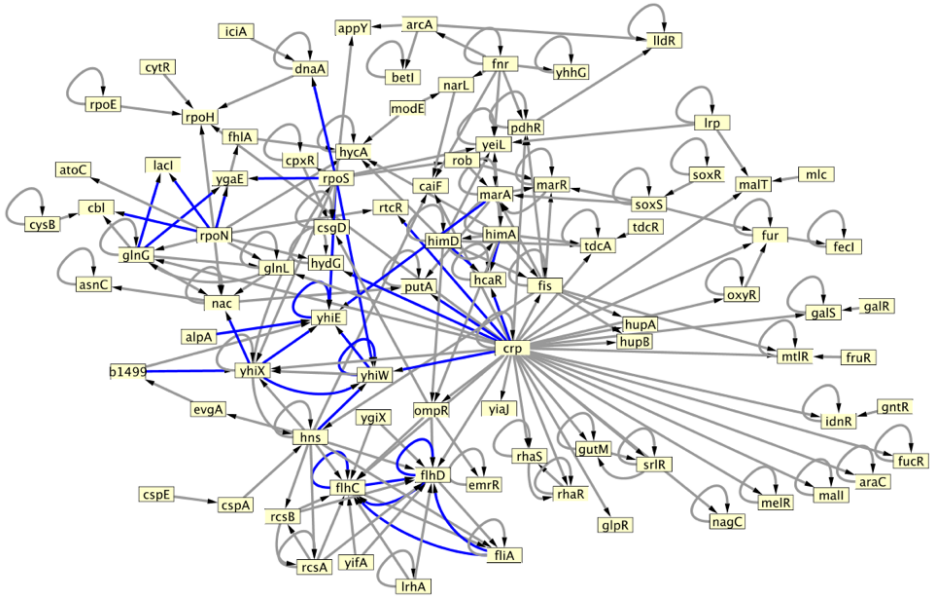
Summary on *de novo* network inference

- Feature selection methods seem to be state-of-the-art
- Performance remains low: recall below 10% for the best-known network
- How to infer the 90% of difficult interactions??
  - improve *de novo* methods
  - change the paradigm

Outline

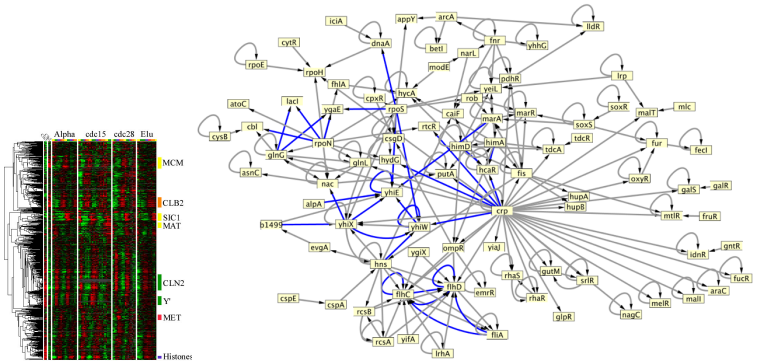
Motivations





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

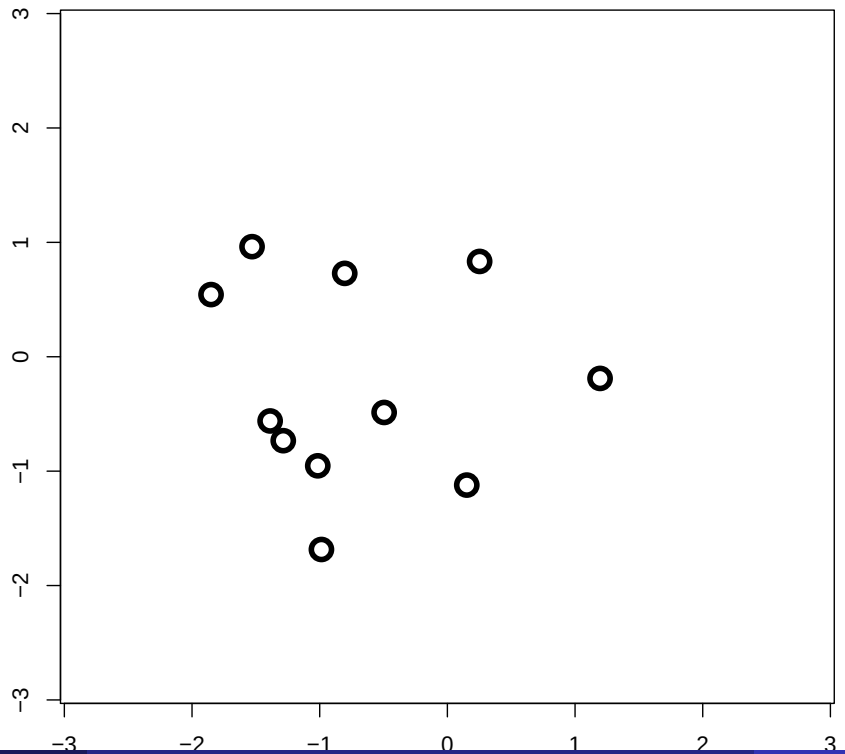
Change of paradigm



- New 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
- Caveats:
  - We need known interactions
  - We may not find completely different interactions from those we know

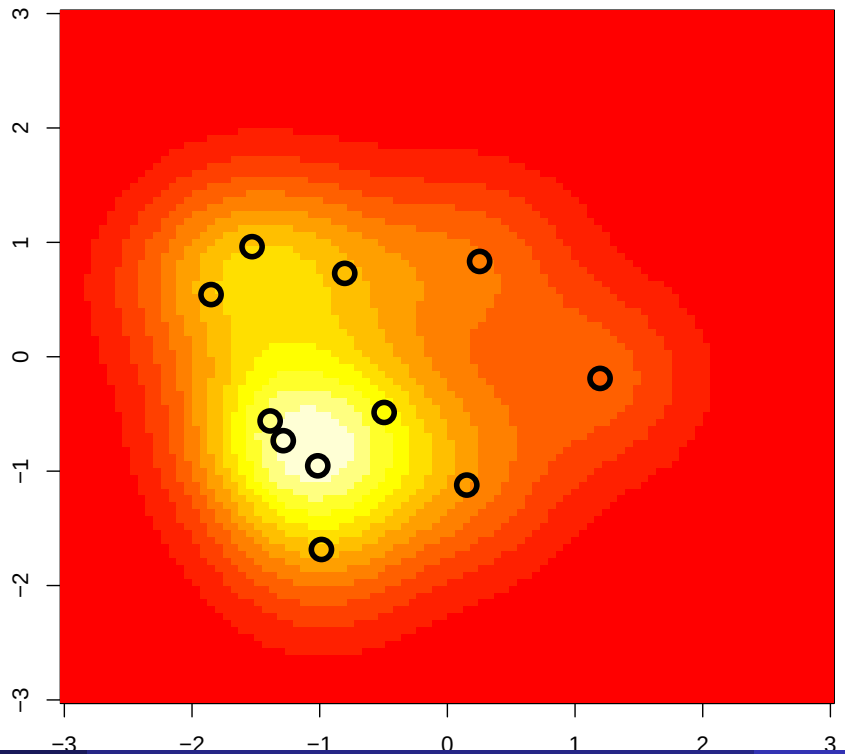
One-class learning approaches

- 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



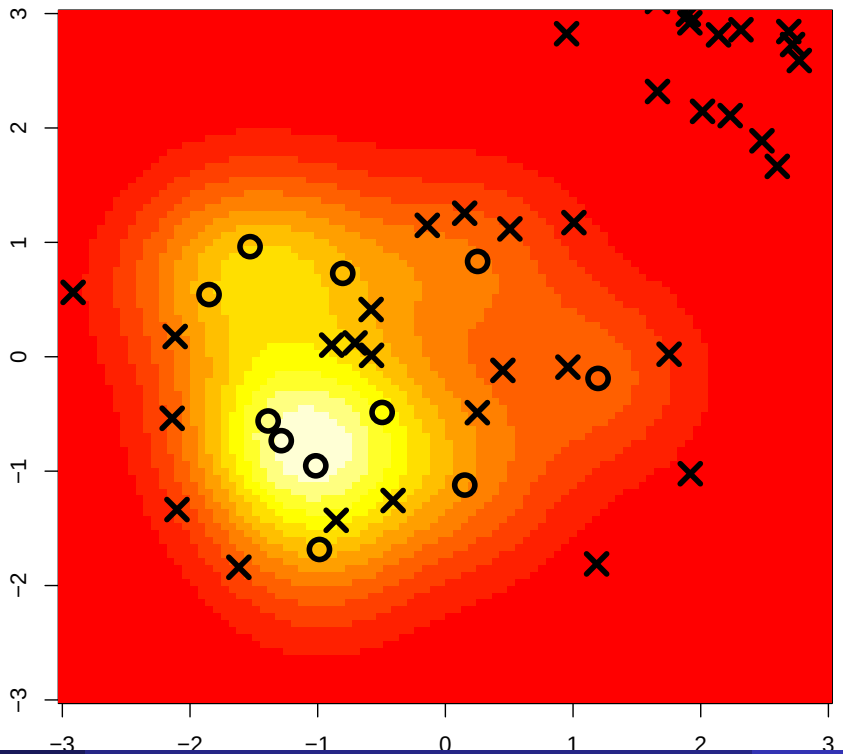
One-class learning approaches

- 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

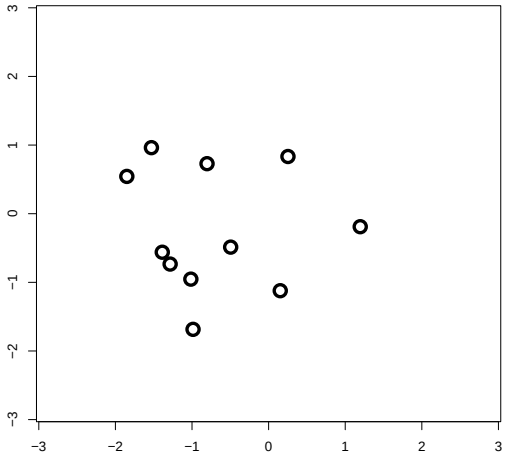


One-class learning approaches

- 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



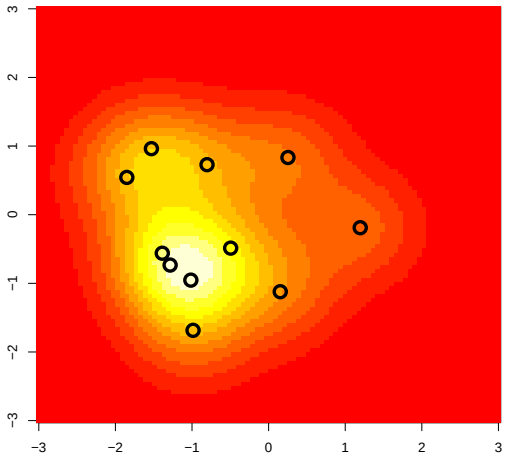
- Kernel density estimation

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

- One-class SVM

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

Estimating the scoring function: examples



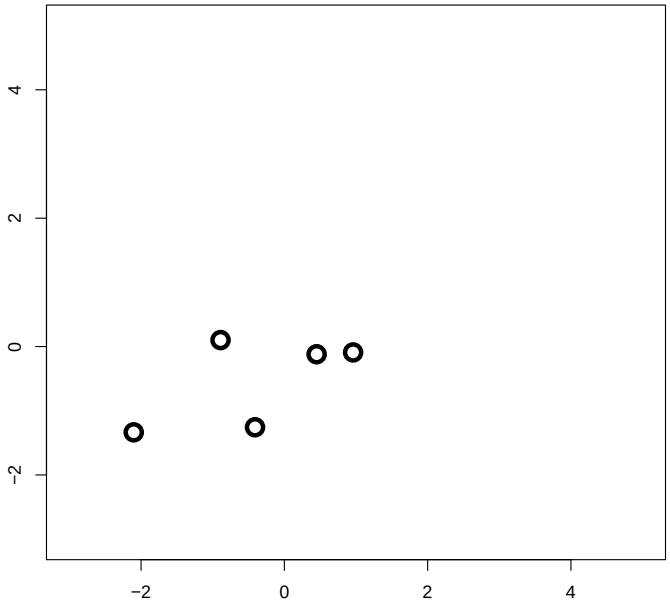
- Kernel density estimation

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

- One-class SVM

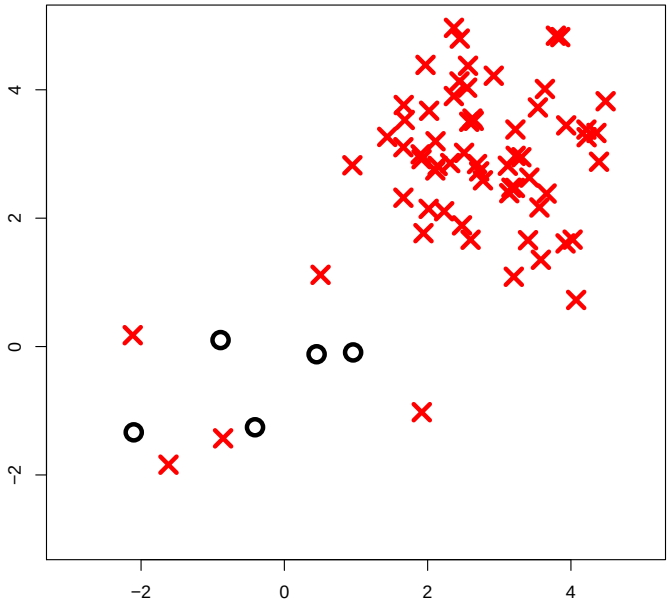
$$s(X) = \sum_{i \in P} \alpha_i \exp\left(-\gamma \| X - X_i \|^2\right)$$

From one-class to PU learning



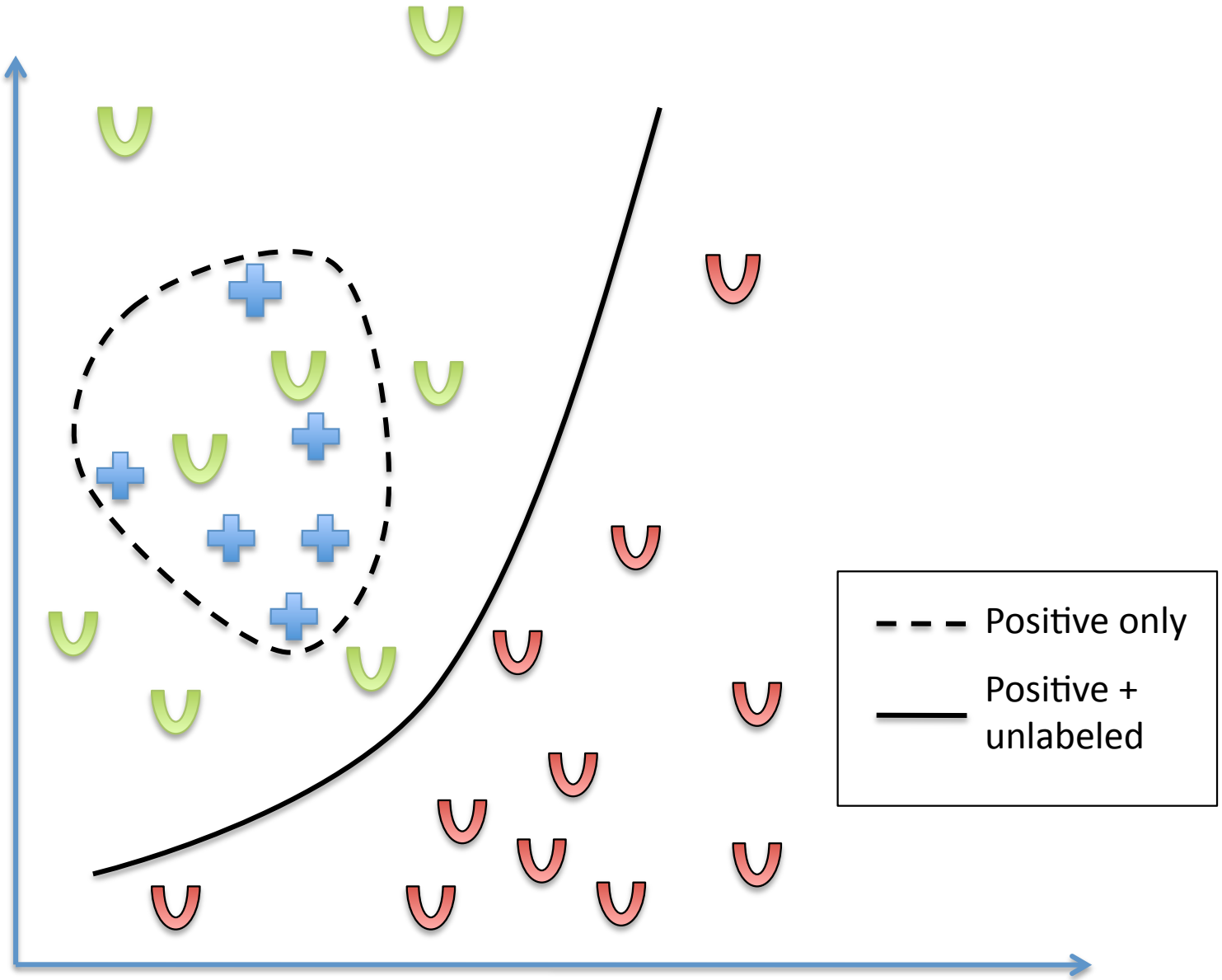
- One class: given genes in P, estimate the function $s(X)$
- PU learning: given genes in P and the set of unlabeled genes U, estimate the scores $s(X_j)$ for $j \in U$

From one-class to PU learning

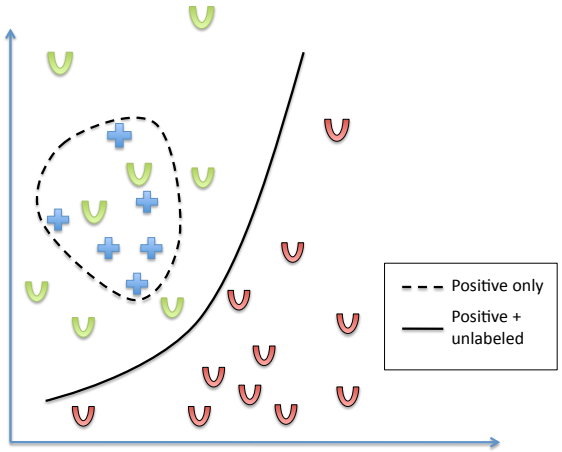


- One class: given genes in P, estimate the function $s(X)$
- PU learning: given genes in P and the set of unlabeled genes U, estimate the scores $s(X_j)$ for $j \in U$

Why PU learning can be better than one-class learning

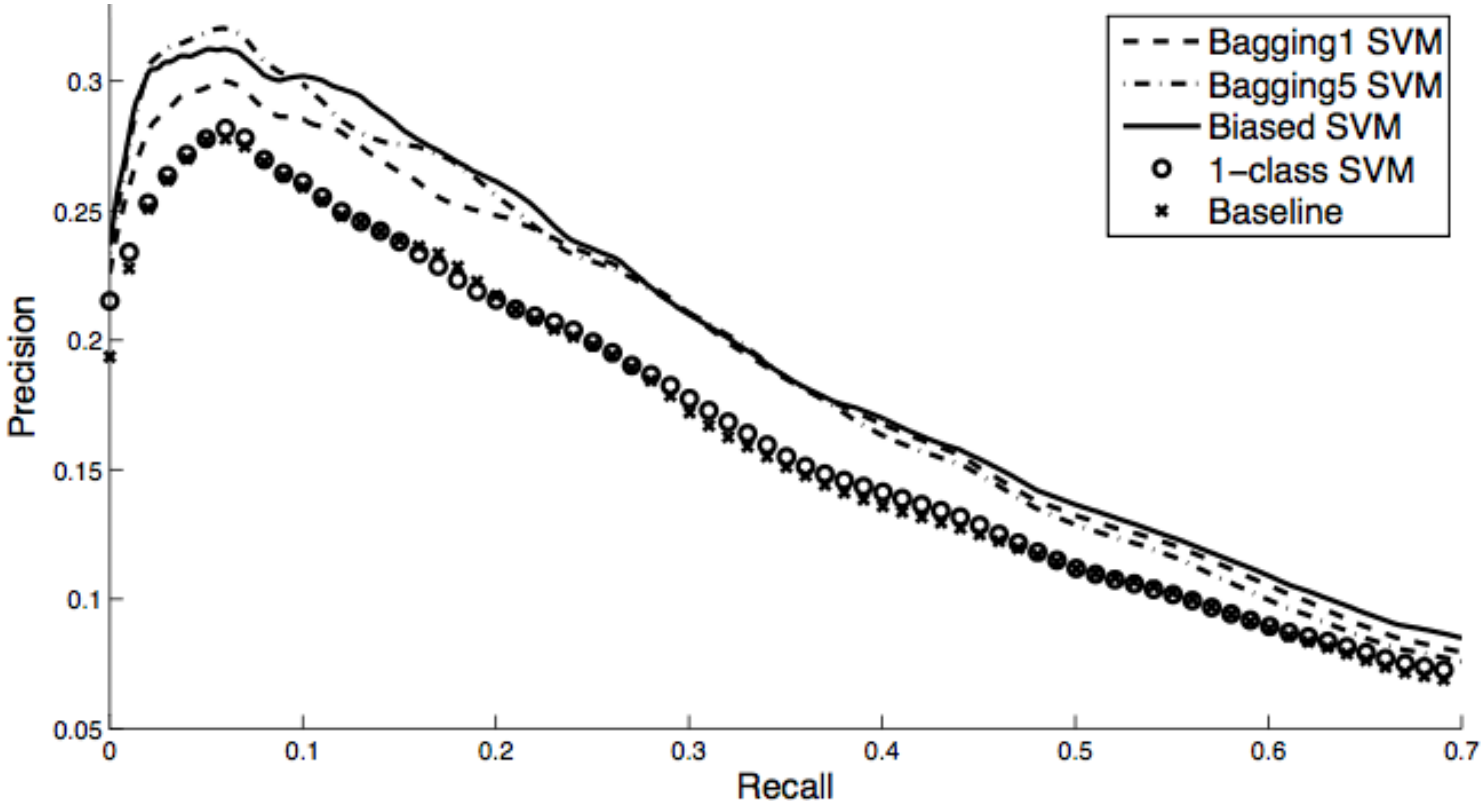


PU learning in practice



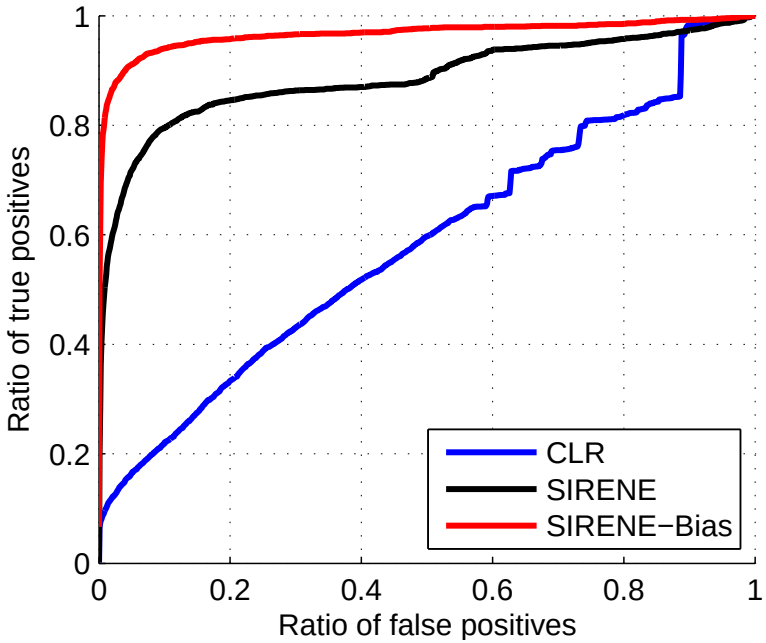
1. Train a classifier to discriminate P from U (eg, SVM or random forest)
2. Rank genes in U by decreasing training score
3. Bagging PU discrimination can help (Mordelet and V., 2010)

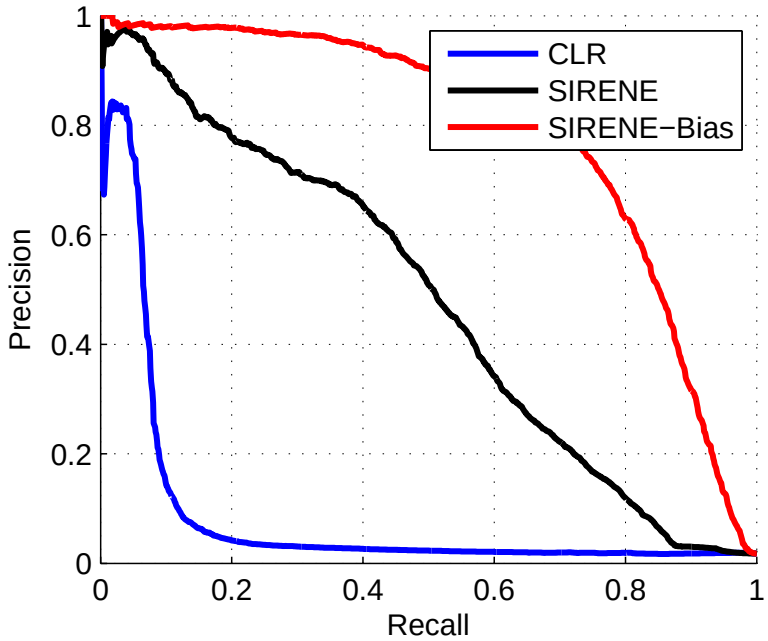
One-class vs PU learning



More in Fantine Mordelet's PhD (2010)

Supervised vs de novo inference

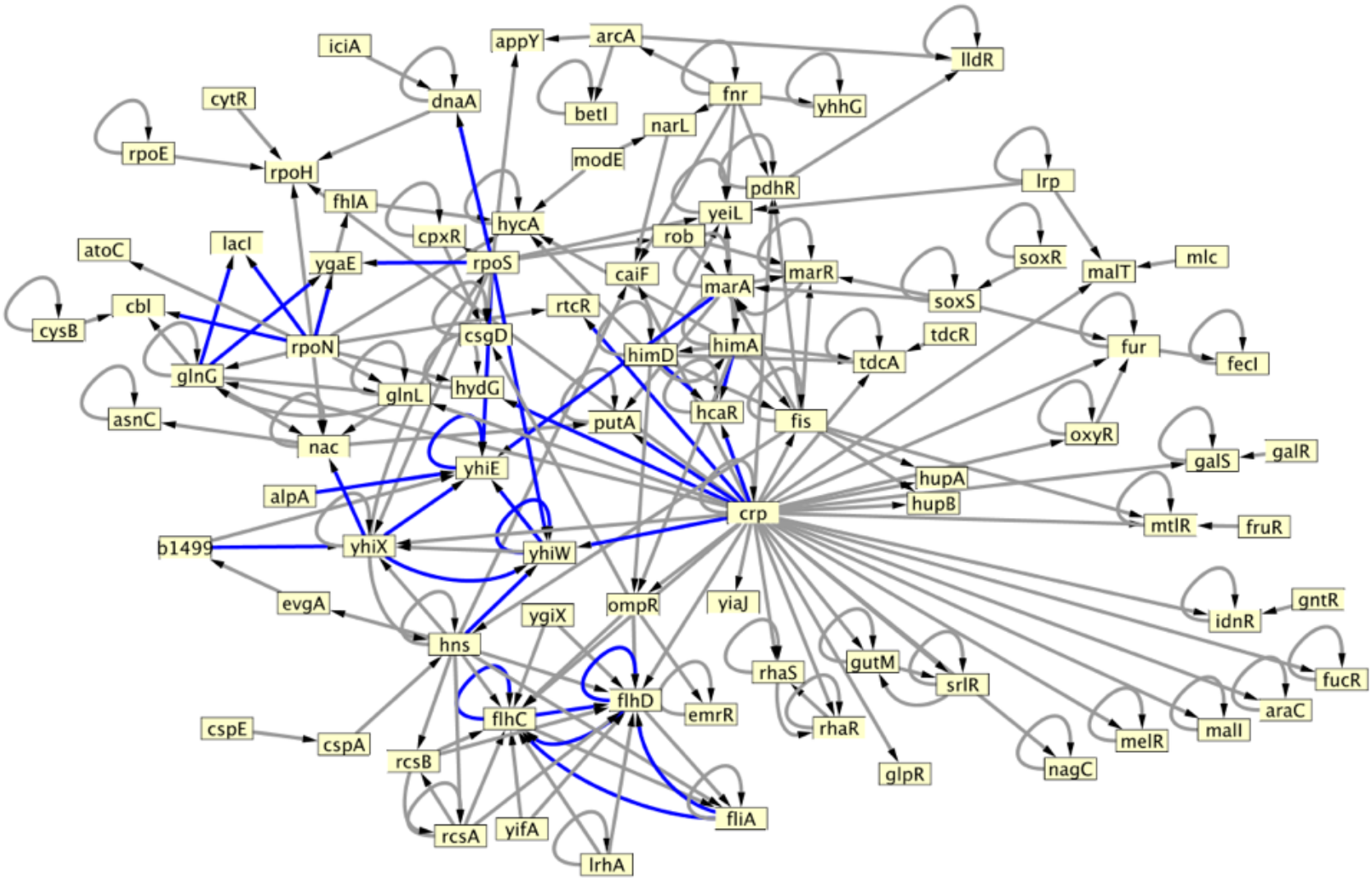




Method	Recall at 60%	Recall at 80%
SIRENE	**44.5%**	**17.6%**
CLR	7.5%	5.5%
Relevance networks	4.7%	3.3%
ARACNe	1%	0%
Bayesian network	1%	0%

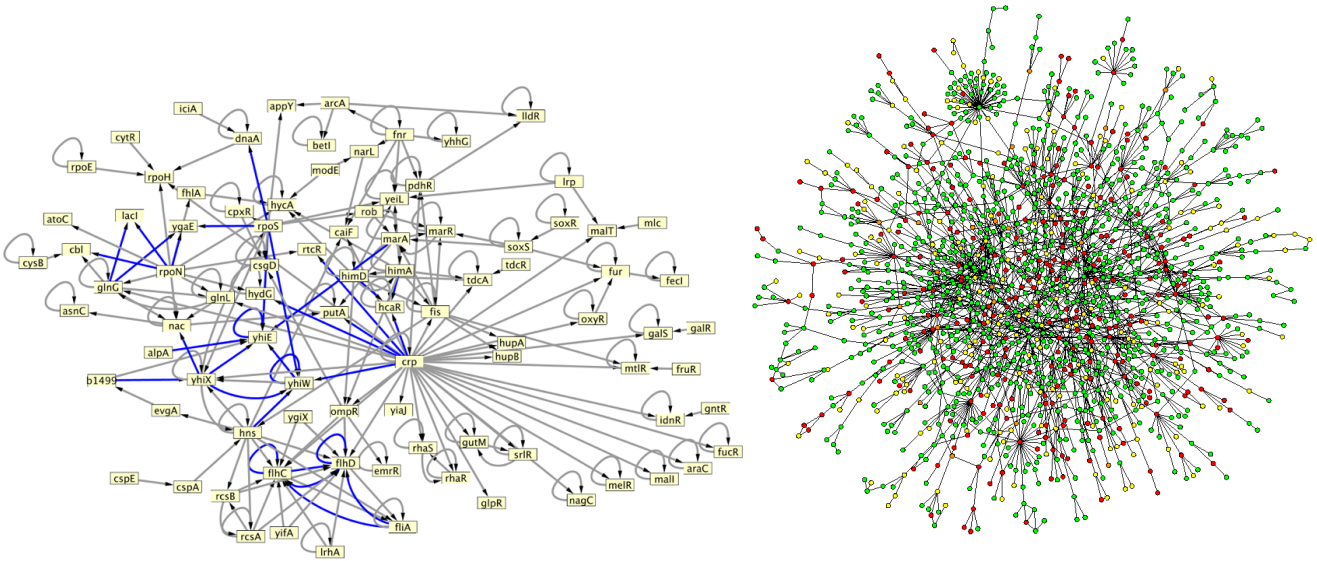
SIRENE = Supervised Inference of REgulatory NEtworks (Mordelet and V., 2008)

Application: predicted regulatory network (E. coli)



Outline

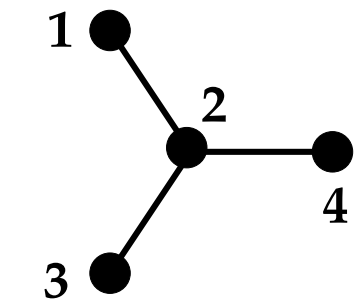
Motivations



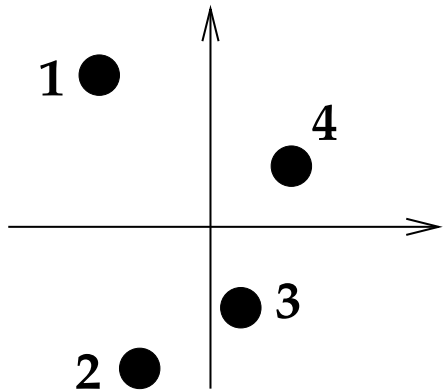
- Local models require enough known targets of each TF. Can we share information across TF?
- For undirected networks (eg, PPI), how to reconcile local predictions?
- Idea: work directly in the space of pairs, to discriminate interacting vs non-interacting pairs.

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!



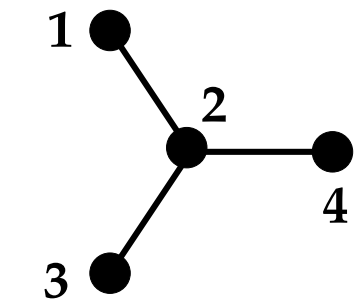
Known graph



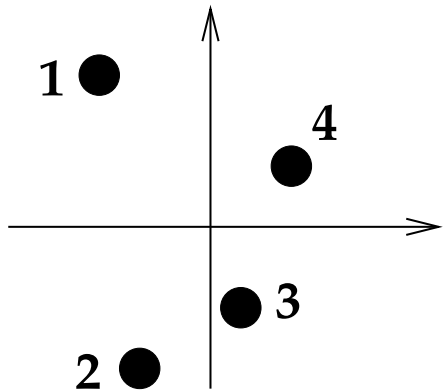
Genomic data

Pattern recognition for pairs: basic issue

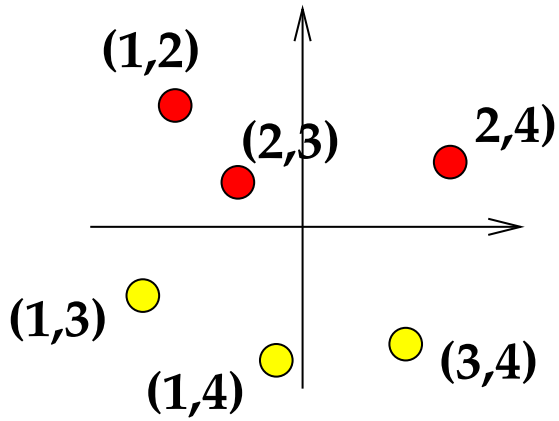
- A pair can be connected (1) or not connected (-1)
- From the known subgraph we can extract examples of connected and non-connected pairs
- However the genomic data characterize individual proteins; we need to work with pairs of proteins instead!



Known graph

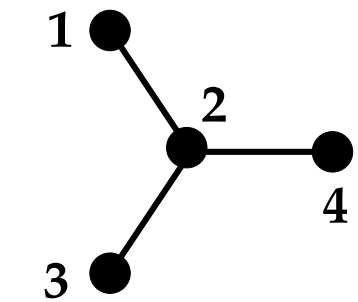


Genomic data

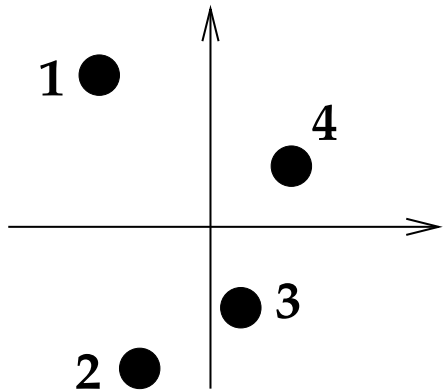


Pattern recognition for pairs: basic issue

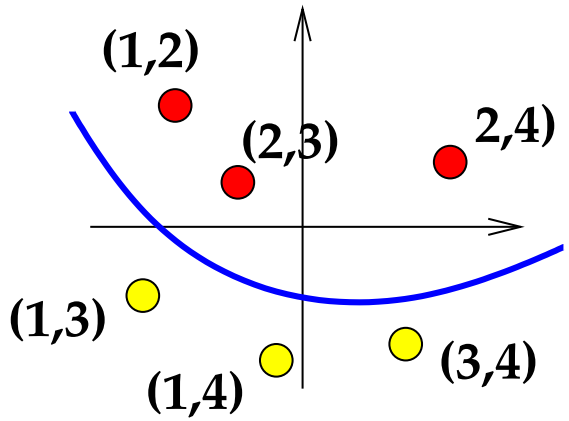
- A pair can be connected (1) or not connected (-1)
- From the known subgraph we can extract examples of connected and non-connected pairs
- However the genomic data characterize individual proteins; we need to work with pairs of proteins instead!



Known graph



Genomic data



Direct sum for ordered pairs?

- Each individual protein is represented by a vector $v \in \mathbb{R}^p$
- How to represent a pair of proteins (u, v) by a vector $\psi(u, v) \in \mathbb{R}^q$?
- 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?

- Each individual protein is represented by a vector $v \in \mathbb{R}^p$
- How to represent a pair of proteins (u, v) by a vector $\psi(u, v) \in \mathbb{R}^q$?
- 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?

- Each individual protein is represented by a vector $v \in \mathbb{R}^p$
- How to represent a pair of proteins (u, v) by a vector $\psi(u, v) \in \mathbb{R}^q$?
- 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: TPPK

- 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:

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

- 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: TPPK

- 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:

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

- 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: TPPK

- 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:

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

- 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 representation: MLPK

- Another symmetric representation:

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

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

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

- Theorem: A SVM with the MLPK kernel trained to discriminate connected from non-connected pairs, solves a metric learning problem (V. et al., 2007)

Technical details

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

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

- Learn the metric so that points close to each other are connected?
- 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:

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

- SVM with MLPK kernel solve it without the constraint $M \geq 0$

Technical details

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

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

- Learn the metric so that points close to each other are connected?
- 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:

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

- SVM with MLPK kernel solve it without the constraint $M \geq 0$

Technical details

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

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

- Learn the metric so that points close to each other are connected?
- 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:

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

- SVM with MLPK kernel solve it without the constraint $M \geq 0$

Technical details

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

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

- Learn the metric so that points close to each other are connected?
- 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:

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

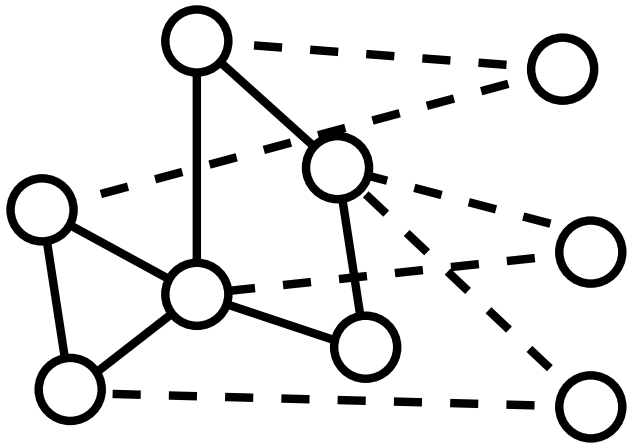
- SVM with MLPK kernel solve it without the constraint $M \geq 0$

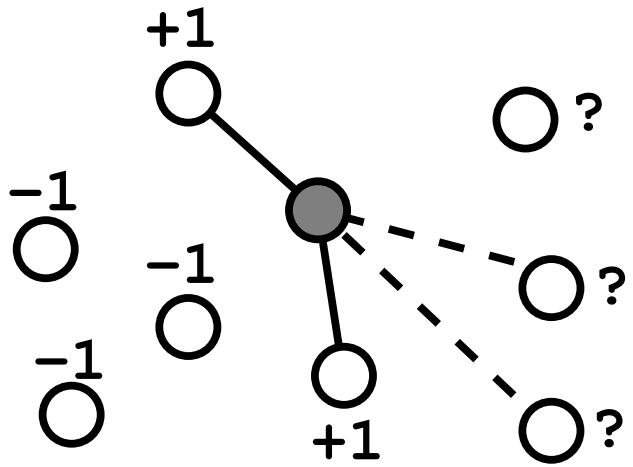
Alternative: symmetrized local models for undirected networks

The idea (Bleakley and V., 2007)

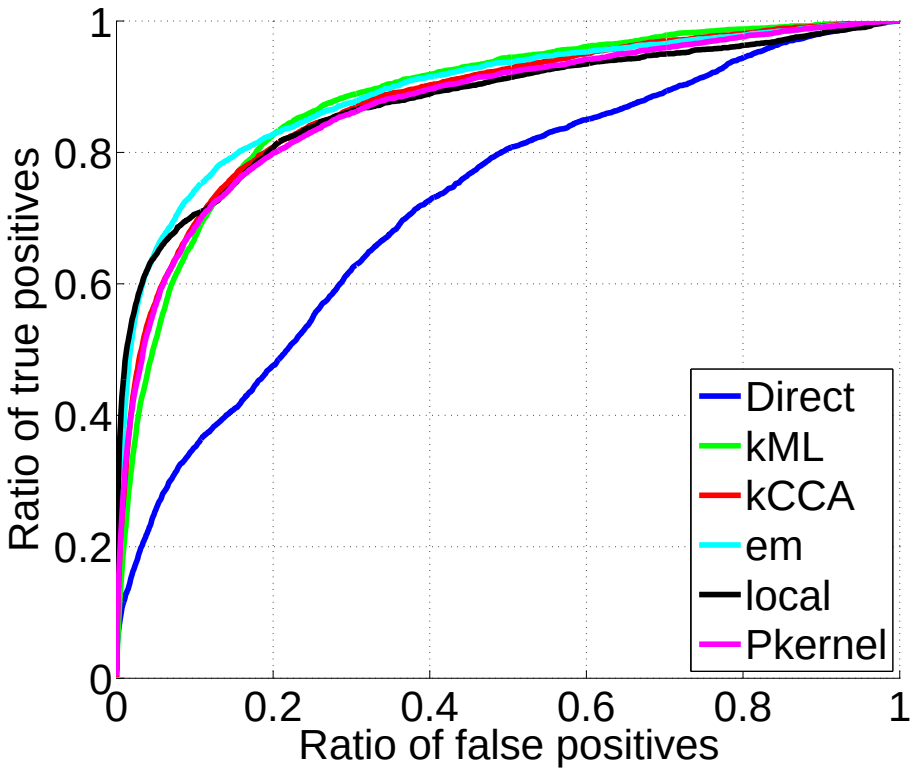
- For each protein P, make a local model using known partners as positive examples to estimate an interaction score $s_P(P')$ for any candidate partner P'
- Symmetrize a posteriori: the interaction score of a candidate pair P, P' is:

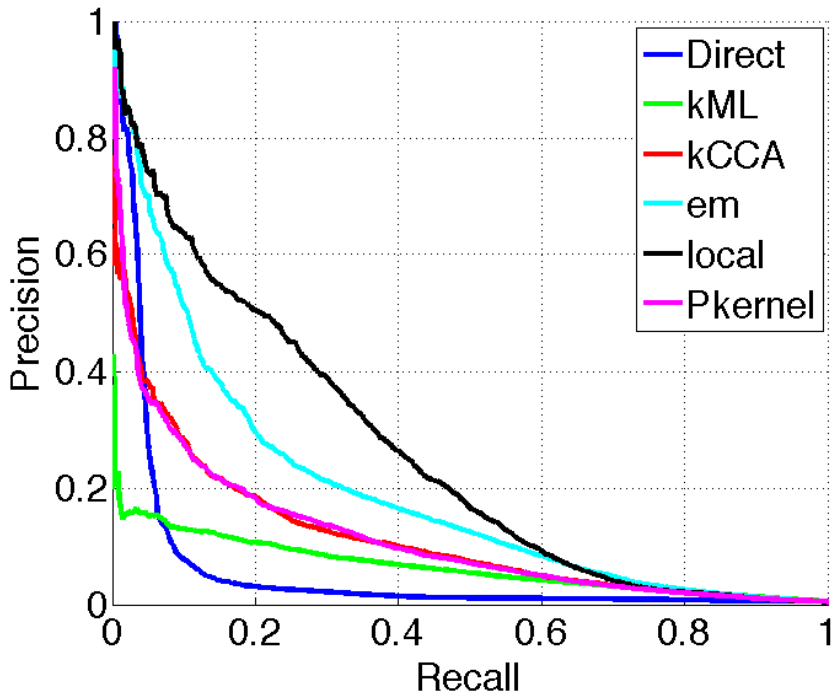
$$s_P(P') + s_{P'}(P)$$





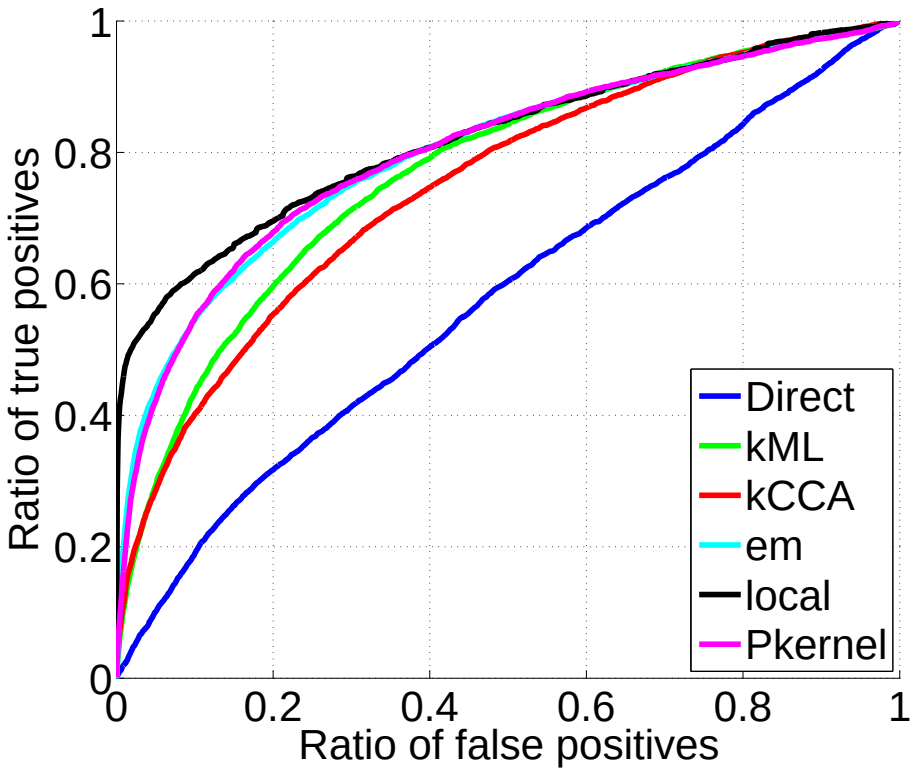
Results: protein-protein interaction (yeast)

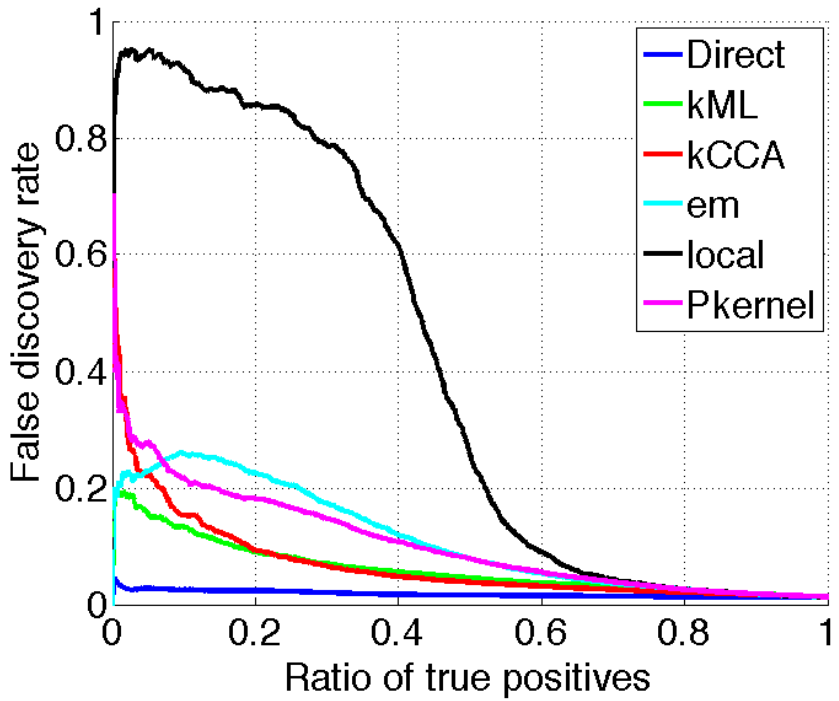




(from Bleakley et al., 2007)

Results: metabolic gene network (yeast)





(from Bleakley et al., 2007)

Outline

Motivation

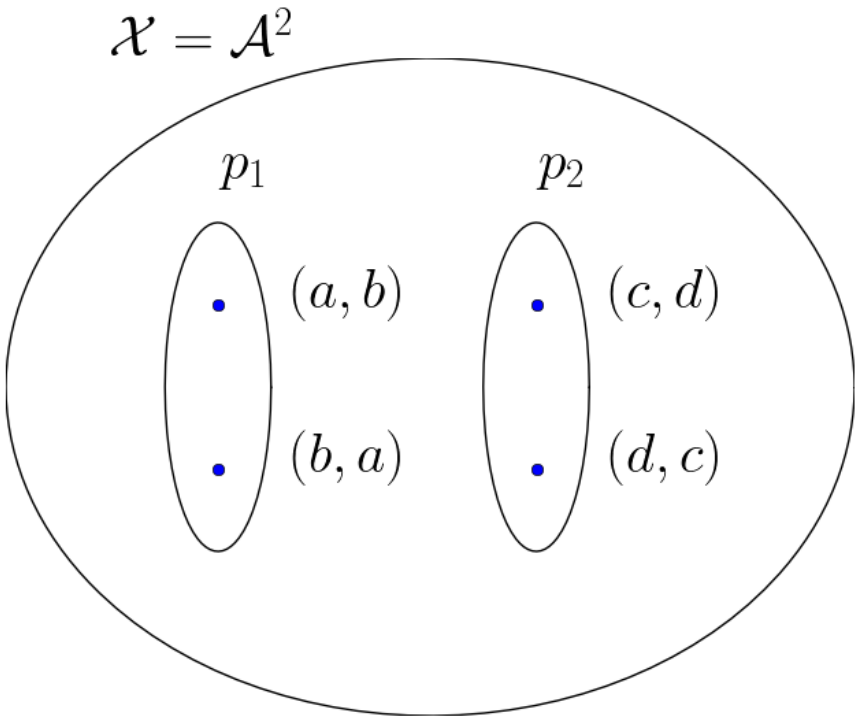
In the case of unordered pairs $\{A, B\}$, pairwise kernels such as the TPPK and local models look very different:

- 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

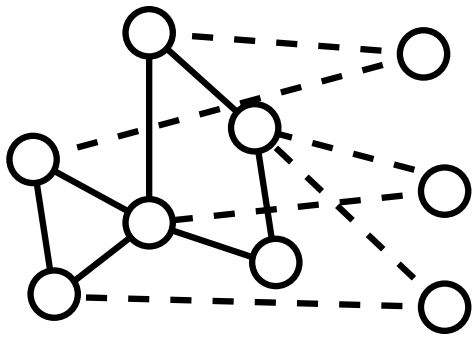
Let $\mathcal{X} = \mathcal{A}^2$ be endowed with the p.d. kernel:

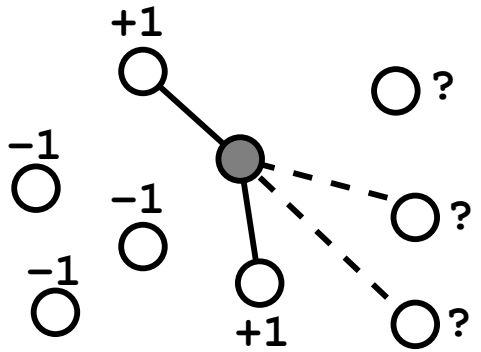
$$K_{\mathcal{X}}\left((a,b),(c,d)\right) = 2K_{\mathcal{A}}(a,c)K_{\mathcal{A}}(b,d)\,. \tag{1}$$

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

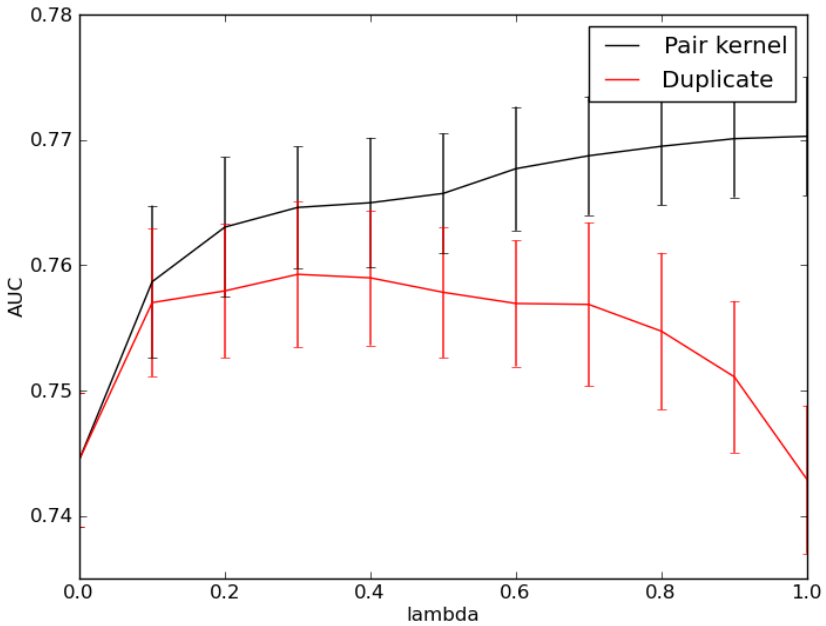
	Strategy 1: pair kernel	Strategy 2: duplication
$K_{\mathcal{X}} = K_{\mathcal{A}} \otimes K_{\mathcal{A}}$	TPPK	TPPK
$K_{\mathcal{X}} = \delta \otimes K_{\mathcal{A}}$	new	Local model

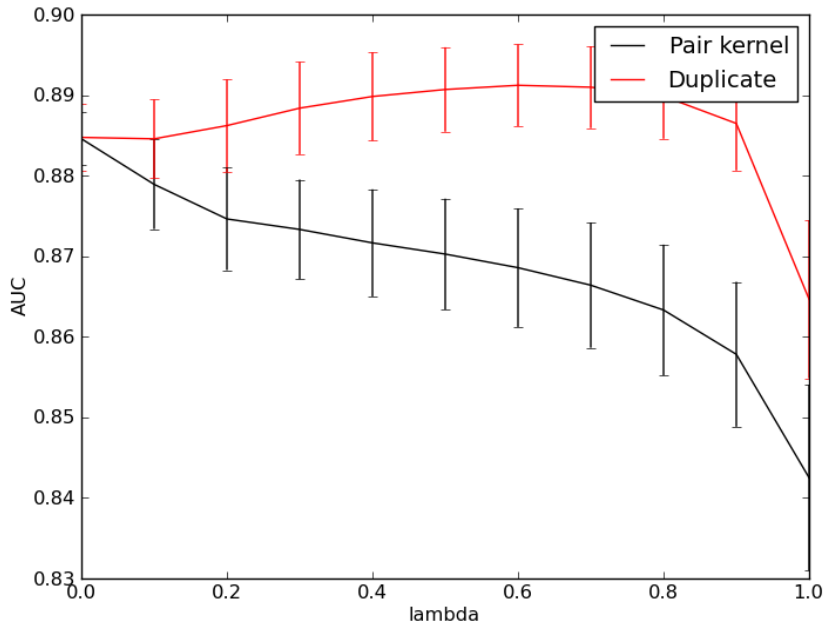
Interpolation:

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

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

Interpolation kernel





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

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, $\lambda = 0$

Outline

Interpolation between local model and TPPK

Take-home messages

- De novo inference: feature selection methods state-of-the-art, but overall performance very limited (recall < 10%)
- Supervised inference: the change of paradigm boosts the performance. Difficult to do better than local models.
- If you already know edges, supervised inference is much more powerful than *de novo* inference

Some interesting questions

- New ideas for *de novo* inference?
  - More direct formulation as structured output learning?
  - Better adjust the complexity of models to the complexity of the task?
- Link *de novo* and supervised inference?
- Combine edge inference with graph models? Link with methods in relational learning and collaborative filtering?

Interpolation between local model and TPPK

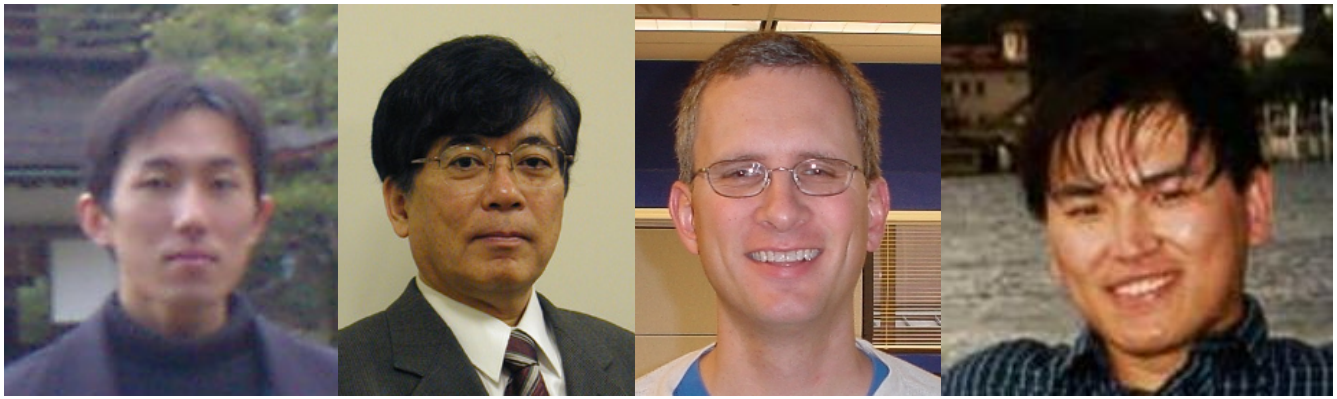
Take-home messages

- De novo inference: feature selection methods state-of-the-art, but overall performance very limited (recall < 10%)
- Supervised inference: the change of paradigm boosts the performance. Difficult to do better than local models.
- If you already know edges, supervised inference is much more powerful than *de novo* inference

Some interesting questions

- New ideas for *de novo* inference?
  - More direct formulation as structured output learning?
  - Better adjust the complexity of models to the complexity of the task?
- Link *de novo* and supervised inference?
- Combine edge inference with graph models? Link with methods in relational learning and collaborative filtering?

Thanks!



Yoshi Yamanishi (ParisTech), Minoru Kanehisa (Kyoto U) Jian Qian, Bill Noble (U Washington), Kevin Bleakley (INRIA), Gerard Biau (ENS Paris), Fantine Mordelet, Martial Hue, Anne-Claire Haury (ParisTech)