Supervised Network inference using Output Kernel Regression

Florence d'Alché–Buc
Joint work with Pierre Geurts*

IBISC CNRS 31 90, Université d'Evry, France
Institut Pasteur, URA CNRS 2172
*GIGA-University of Liege
Email: florence.dalche@ibisc.fr

How to learn biological networks from data ?

- **Predictive approaches** : predict (only) edges in an unsupervised or supervised way [scale (10^3) nodes]
- **Modeling approaches** : model the network as a complex system, can be used to simulate and predict the network behavior ($scale : 10 to 10^2$)]

Learning (biological) networks

Data

- global data
- intervention data

Available knowledge

- edges distribution
- dynamics

labeled data, partial network

Modeling the behavior of the network
Unsupervised
Bayesian networks
Dynamical Bayesian networks
And state-space models
Ordinary Differential Equations
Stochastic Differential Equations

Predicting edges (only)
unsupervised
Gaussian Graphical models

Supervised
Decision trees, SVM,ILP
Metric and kernel learning

Outline

Supervised learning of edges

- Directed edges : case of transcriptional regulatory network
- Non directed edges: case of protein-protein interaction network

Supervised learning of directed edges

Setting of the problem

- Training sample $S = \{(w_i = (v_i, v_i'), y_i), i = 1...n\}$ where w_i are **couples** of genes descriptors v_i and v_i' (think transcription factor and potential regulee)
- $y_i \in \{0, 1\}$ indicates if there is v_i is a transcription factor for v_i'.
- **Goal** : from training data , learn a classification function able to predict if an edge exists from a pair of inputs.
- **First reference** : Qian et al. 2003, Bioinformatics, with SVM.
- But you can use your preferred classifier

Supervised prediction of non directed edges

- **Training sample** $LS = \{(w_i = (v_i, v_i'), y_i), i = 1...n\}$ where w_i are couples of components v_i and v_i' (think proteins)
- $y_i \in Y$ indicates if there is an edge or not between v_i and v_i'.
- Noble et al. in 2005 (SVM) with kernel combination
- Further studied by Biau and Bleakley 2006, Bleakley, Biau and Vert, 2007

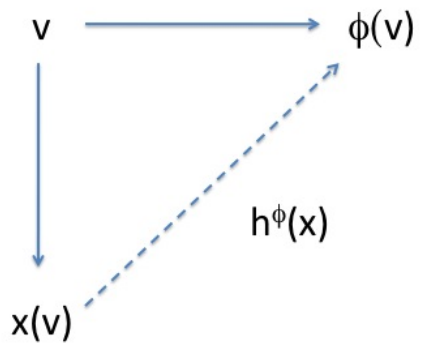
Supervised prediction of non oriented edges by similarity learning

- In the case of non oriented graphs, a similarity between components can be learnt instead of a classification function (Yamanishi and Vert 2005)
- Our proposal: see the task as kernel learning with a new kind of regression:output kernel regression (Geurts et al. 2006,2007, 2009 in prep.)

Supervised Learning with output feature space

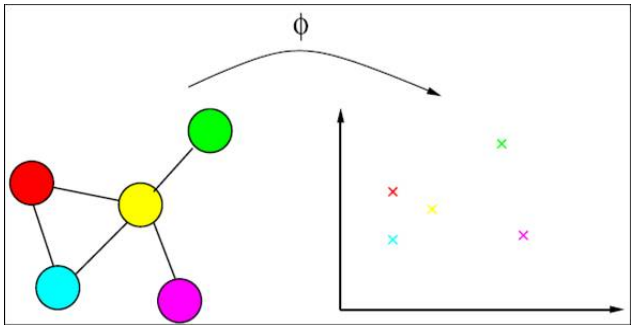
- Suppose we have a learning sample $LS = \{x_i = x(v_i), i = 1, \ldots, N\}$ drawn from a fixed but unknown probability distribution
- with an additional information provided by a Gram matrix $K = k_{ij} = k(v_i, v_j)$, for$i, j = 1, \ldots, N$ that expresses how much objects $v_i, i = 1...n$ are close to each other.
- Let us call respectively ϕ the implicit output feature map corresponding to k a positive definite kernel defined on $\mathcal{V} \times \mathcal{V}$ such that $< \phi(v), \phi(v') >= k(v, v')$.

Supervised Learning with output feature space



From a learning sample $\{(x_i, K_{ij} | i = 1, \ldots, N, j = 1, \ldots, N\}$ *with* $x_i \in \mathcal{X}$*,find a function* $f : \mathcal{X} \rightarrow \mathcal{F}_K$ *that minimizes the empirical mean of some loss function* $\ell : \mathcal{F}_k \times \mathcal{F}_K \rightarrow \mathbb{R}$ *(possibly with an additional regularization term)*

Supervised inference of edges in a graph 1



- For proteins $v_1, ..., v_N$, we have : feature vectors $x(v_i), i = 1...N$
- Let's assume that for these proteins we know the protein-protein interaction graph
- Let's define a Gram matrix K defined as $K_{i,j} = k(v_i, v_j)$ that reflects the proximity between proteins v, as vertices in this graph.
- What does it means ?
- We can deal with vertices in the graph as vectors in the vectorial space spanned by the $\phi(v_i)$.

Supervised inference of edges in a graph 2

- Define a new learning method that can infer a function $h^\phi : \mathcal{X} \rightarrow \mathcal{F}_K$ to get for a given $x(v)$, an approximation $h^\phi \in \mathcal{F}_K$ of $\phi(v)$
- This proxy will be used to get an approximation $g(x(v), x(v')) = \langle h^\phi(x(v)), h^\phi(x(v')) \rangle$ of the kernel value between v and v' described by their input feature vectors $x(v)$ and $x(v')$
- By construction g is a kernel
- Connect these two vertices if $g(x(v), x(v')) > \theta$
- by varying θ we get different tradeoffs between true positive and false positive rates)

A kernel on graph nodes

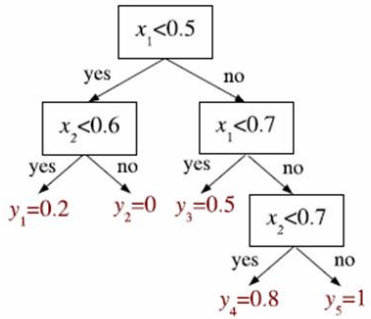
- Given an adjacency matrix, define a diffusion kernel (Kondor and Lafferty, 2002) as:
  The Gram matrix K with $K_{i,j} = k(v_i, v_j)$ is given by:

$$K = \exp(-\beta L)$$

  where the graph Laplacian L is defined by:

$$L_{i,j} = \begin{cases} d_i & \text{the degree of node } v_i \text{ if } i = j; \\ -1 & \text{if } v_i \text{ and } v_j \text{ are connected;} \\ 0 & \text{otherwise.} \end{cases}$$

Standard regression trees



- A learning algorithm that solves the regression problem for one-dimensional output with a tree structured model
- Basic idea of the learning procedure:
  - Recursively split the learning sample with tests based on the inputs trying to reduce as much as possible the variance of the output
  - Stop when the output is constant in the leaf (or some stopping criterion is met)

Regression trees on multiple outputs

- The best split is the one that maximizes the empirical variance reduction on current training data:

$$\text{Score}_R(\mathit{Test}, S) = \text{var}\{y|S\} - \frac{N_l}{N}\text{var}\{y|S_l\} - \frac{N_r}{N}\text{var}\{y|S_r\},$$

where N is the size of S, N_l (resp. N_r) the size of S_l (resp. S_r), and $\text{var}\{Y|S\}$ denotes the variance of the output Y in the subset S:

- $$\text{var}\{y|S\} = \frac{1}{N}\sum_{i=1}^{N} ||y_i - \overline{y}||^2 \text{ with } \overline{y} = \frac{1}{N}\sum_{i=1} y_i$$

  which is the average distance to the center of mass (or the average variance over all outputs).
- Predictions at leaf nodes are the centers of mass

$$\frac{1}{N_L}\sum_{i=1}^{N_L} y_i$$

Regression trees in output feature space

- Given a kernel k on the output space (with corresponding feature map ϕ, the idea is to grow a multiple output regression tree in the output feature space $\mathcal{F}_{||}$:
  - The variance becomes:

$$\text{var}\{\phi(v)|S\} = \frac{1}{N}\sum_{i=1}^{N}||\phi(v_i) - \frac{1}{N}\sum_{i=1}\phi(v_i)||^2$$

  - Using kernel trick, we have:

$$\text{var}\{\phi(v)|S\} == \frac{1}{N}\sum_{i=1}^{N}k(v_i, v_i) - \frac{1}{N^2}\sum_{i,j=1}^{N}k(v_i, v_j)$$

  - Predictions at leaf nodes are still the average of output data:

$$\hat{\phi}_L = \frac{1}{N_L}\sum_{i=1}^{N_L}\phi(v_i)$$

Kernelization

- The variance may be written:

$$\begin{aligned} \text{var}\{\phi(v)|S\} &= \frac{1}{N}\sum_{i=1}^{N}||\phi(v_i) - \frac{1}{N}\sum_{i=1}\phi(v_i)||^2 \\ &= \frac{1}{N}\sum_{i=1}^{N} < \phi(v_i), \phi(v_i) > - \frac{1}{N^2}\sum_{i,j=1}^{N} < \phi(v_i), \phi(v_j) >, \end{aligned}$$

  which makes use only of dot products between vectors in the output feature space
- We can use the kernel trick and replace these dot-products by kernels:

$$\text{var}\{\phi(v)|S\} = \frac{1}{N}\sum_{i=1}^{N} k(v_i, v_i) - \frac{1}{N^2}\sum_{i,j=1}^{N} k(v_i, v_j)$$

- From kernel values only, we can thus grow a regression tree that minimizes output feature space variance

Improvements by Ensemble methods with Output Kernel trees

- Trees can be improved if combined linearly in random forests (ET,...), in bagging and boosting methods
- All ensemble methods can be extended to deal with Output kernel regression
- Drawback: loss of interpretability, still the importance of each feature can be computed easily

Gradient boosting with square loss (Friedman 1999

- If $\ell(y_1, y_2) = (y_1 - y_2)^2/2$, the algorithm becomes:

LS Boost

1. $F_0(x) = \frac{1}{N} \sum_{i=1}^{N} y_i$
2. For $m = 1$ to M do:
   1. $y_i^m = y_i - F_{m-1}(x_i),\ i = 1, \ldots, N$ (compute the residuals)
   2. $a_m = \arg\min_a \sum_{i=1}^{N} (y_i^m - h(x_i; a))^2$ (fit them)
   3. $F_m(x) = F_{m-1}(x) + h(x; a_m)$ (update the function)

- e.g., $h(x; a)$ are small regression trees (Friedman's Multiple Additive Regression Trees, MART).
- In practice, it is very useful to use a shrinkage parameter ($\nu << 1$) to control the learning rate

Gradient boosting with square loss (Friedman 1999

- If $\ell(y_1, y_2) = (y_1 - y_2)^2/2$, the algorithm becomes:

LS Boost

1. $F_0(x) = \frac{1}{N}\sum_{i=1}^{N} y_i$
2. For $m = 1$ to M do:
   1. $y_i^m = y_i - F_{m-1}(x_i),\ i = 1, \ldots, N$ (compute the residuals)
   2. $a_m = \arg\min_a \sum_{i=1}^{N} (y_i^m - h(x_i; a))^2$ (fit them)
   3. $F_m(x) = F_{m-1}(x) + \nu h(x; a_m)$ (update the function)

- e.g., $h(x; a)$ are small regression trees (Friedman's Multiple Additive Regression Trees, MART).
- In practice, it is very useful to use a shrinkage parameter ($\nu << 1$) to control the learning rate

Kernelizing the output

LS Boost in a kernelized output space

1. $F_0^\phi(x) = \frac{1}{N}\sum_{i=1}^N \phi(y_i)$
2. For $m = 1$ to M do:
   1. $\phi_i^m = \phi(y_i) - F_{m-1}^\phi(x_i), i = 1, \ldots, N$ (compute the residuals)
   2. $a_m = \arg\min_a \sum_{i=1}^N ||\phi_i^m - h^\phi(x_i; a)||^2$ (fit them)
   3. $F_m^\phi(x) = F_{m-1}^\phi(x) + h^\phi(x; a_m)$ (update the function)

- Replace y by a vector $\phi(y)$ from some feature space $\mathcal{H}$ (in which we only assume it is possible to compute dot-products)
- F^ϕ and h^ϕ are now functions from $\mathcal{X}$ to $\mathcal{H}$
- Minimizes square error in $\mathcal{H}$:

$$E_{x,y}\{||\phi(y) - F^\phi(x)||^2\}$$

Kernelizing the output

LS Boost in a kernelized output space

1. $F_0^{\phi}(x) = \frac{1}{N}\sum_{i=1}^{N}\phi(y_i)$
2. For $m = 1$ to M do:
   1. $\phi_i^m = \phi(y_i) - F_{m-1}^{\phi}(x_i), i = 1, \ldots, N$ (compute the residuals)
   2. $a_m = \arg\min_a \sum_{i=1}^{N} ||\phi_i^m - h^{\phi}(x_i; a)||^2$ (fit them)
   3. $F_m^{\phi}(x) = F_{m-1}^{\phi}(x) + h^{\phi}(x; a_m)$ (update the function)

- To be a feasible solution, we need to be able to compute from kernel only:
  - the output Gram matrix K^m at step m, i.e. $K_{i,j}^m = \langle \phi_i^m, \phi_j^m \rangle$ (*to compute 2.2*)
  - $\langle F_M^{\phi}(x), \phi(y) \rangle, \forall x, y$ (*to compute predictions, pre-images*)
- This is possible when $h^{\phi}(x; a_m)$ at step m may be written

$$h^{\phi}(x; a_m) = \sum_{i=1}^{N} w_i(x; a_m)\phi_i^m$$

Results

- Application to two networks in the Yeast:
  - Protein-protein interaction network: 984 proteins, 2478 edges (Kato et al., 2005)
  - Enzyme network: 668 enzymes and 2782 edges (Yamanishi et al., 2005)
- Input features:
  - Expression data: expression of the gene in 325 experiments
  - Phylogenetic profiles: presence or absence of an ortholog in 145 species
  - Localization data: presence or absence of the protein in 23 intracellular location
  - Yeast two hybrid data: data from a high-throughput experiment to detect protein-protein interactions

Comparison with full kernel based methods

Protein network

Inputs	OK3+ET	[1]
expr	0.851	0.776
phy	0.693	0.767
loc	0.725	0.788
y2h	0.790	0.612
All	**0.910**	**0.939**

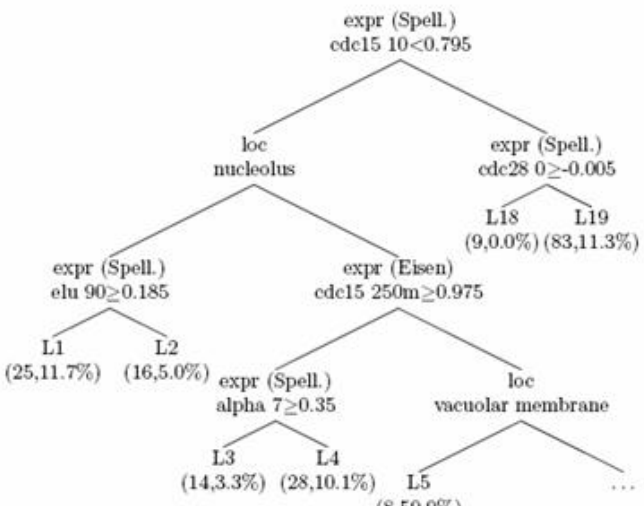
Enzyme network

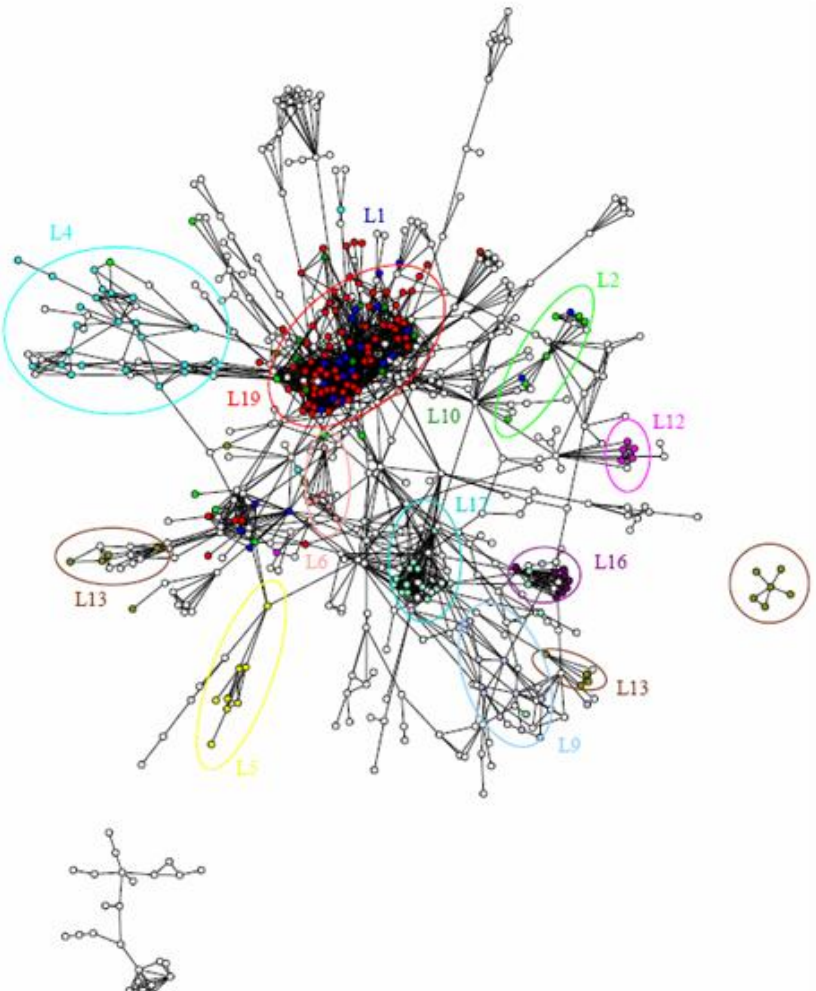
Inputs	OK3+ET	[2]
expr	0.714	0.706
phy	0.815	0.747
loc	0.587	0.577
All	**0.847**	**0.804**

- [1] Kato et al. : EM based algorithm for kernel matrix completion
- [2] Yamanishi et al. : compare a kernel canonical correlation analysis based solution and a metric learning approach

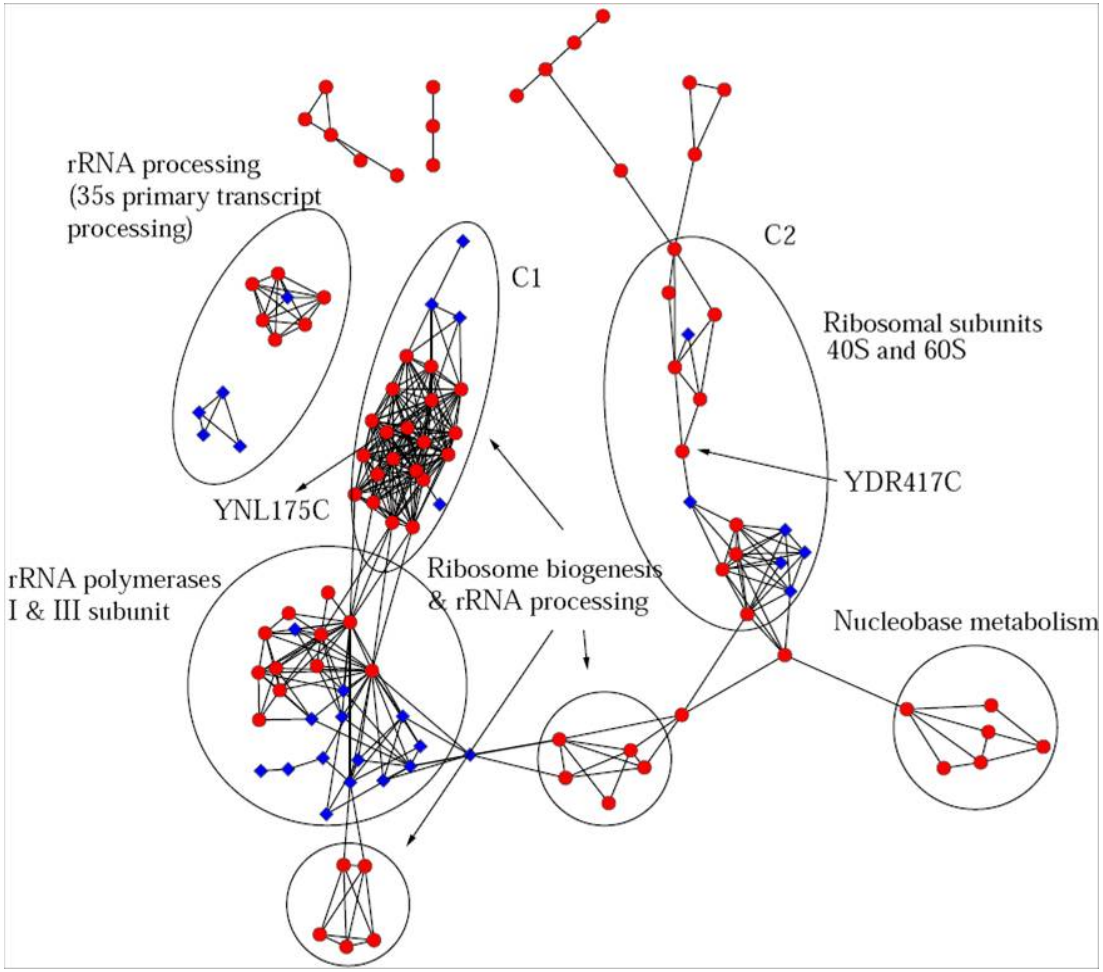
Interpretability: rules and clusters (an example with a protein-protein network)

#	Att.	Imp
1	phy - dre	0.011
2	phy - rno	0.009
3	expr (Eisen) - cdc15 120m	0.008
4	phy - ecu	0.008
5	expr (Eisen) - cdc15 160m	0.008
6	phy - pfa	0.007
7	phy - mmu	0.007
8	loc - cytoplasm	0.006
9	expr (Eisen) - cdc15 30m	0.005
10	expr (Eisen) - elutriation 5.5hrs	0.005





Network completion and function prediction for yeast data



Challenges in supervised predictive approaches

- Transductive learning (current work : completion of the protein-protein interaction network around CFTR protein (cystic fibrosis) with A. Edelman, Necker)
- Issue : unbalanced distribution of positive and negative examples
- Change cost functions
- Interpret learning in feature output space as an interpolation problem (find a surrogate function of the kernel)