Prediction in kernelized output spaces: output kernel trees and ensemble methods

Pierre Geurts    Florence d’Alché–Buc

IBISC CNRS, Université d’Evry, GENOPOLE, Evry, France
Department of EE and CS, University of Liège, Belgium

25 janvier 2007
Motivation

- In many domains (e.g. text, computational biology), we want to predict complex or structured outputs. e.g. graphs, time series, classes with hierarchical relations, position in a graph, images...
- The main goal of our research team is to develop machine learning tools to extract structures:
- We try to address this issue by several ways and through text and systems biology applications:
  - learning structure of BN (Bayesian networks) and DBN (Dynamic bayesian network) : unsupervised approaches
  - learning interactions as a classification concept : supervised and semi-supervised approaches
  - learning mapping between structures when input and output are strongly dependent : supervised approaches
  - learning mapping between input feature vectors and structured outputs (this talk)
Supervised learning with structured outputs

**Example 1**: Image reconstruction

**Example 2**: Find the position of a gene/protein/enzyme in a biological network from various biological descriptors (function of the protein, localization, expression data)

Very few solutions exist for these tasks (one precursor: KDE), none are explanatory

We present a set of methods for handling complex outputs that have some explanatory power and illustrate it on these two problems with a main focus on the biological network completion

- Output Kernel Tree: an extension of regression tree to kernelized output spaces
- Ensemble methods devoted to regressors in kernelized output spaces
Outline

1 Motivation
2 Supervised learning in kernelized output spaces
3 Output Kernel Tree
4 Ensemble methods
   - Parallel ensemble methods
   - Gradient boosting
5 Experiments
   - Image reconstruction
   - Completion of biological networks
   - Boosting
6 Conclusion and future works
Supervised learning with complex outputs

- Suppose we have a sample of objects \( \{o_i, i = 1, \ldots, N\} \) drawn from a fixed but unknown probability distribution.
- Suppose we have two representations of the objects:
  - an input feature vector representation: \( x_i = x(o_i) \in \mathcal{X} \)
  - an output representation: \( y_i = y(o_i) \in \mathcal{Y} \) where \( \mathcal{Y} \) is not necessary a vectorial space (it can be a finite set with complex relations between elements).

From a learning sample \( \{(x_i, y_i)|i = 1, \ldots, N\} \) with \( x_i \in \mathcal{X} \) and \( y_i \in \mathcal{Y} \), find a function \( h: \mathcal{X} \rightarrow \mathcal{Y} \) that minimizes the expectation of some loss function \( \ell: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R} \) over the joint distribution of input/output pairs:

\[
E_{x,y}\{\ell(h(x), y)\}
\]

- Complex outputs: no constraint (for the moment) on the nature of \( \mathcal{Y} \).
General approach

Object $o_i$ $\rightarrow$ $y(o_i) = y_i$

$x$

$x(o_i) = x_i$
Use the kernel trick for the outputs

- Additional information to the training set:
  - A Gram matrix $K = (k_{ij})$, with $k_{ij} = k(y_i, y_j)$ and $k$ a Mercer Kernel with the corresponding mapping $\phi$ such that $k(y, y') = <\phi(y), \phi(y')>$.

- Approach:
  1. Approximate the feature map $\phi$ with a function $h^\phi : X \rightarrow H$ defined on the input space
  2. Get a prediction in the original output space by approximating the function $\phi^{-1}$ (pre-image problem)
Possible applications

- Learning a mapping from an input vector into a structured output (graphs, sequences, trees, time series...)
- Learning with alternative loss functions (hierarchical classification for instance)
- Learning a kernel as a function of some inputs
In some applications, we want to learn a relationship between objects rather than an output (e.g. network completion problem).

Learning data set: \( \{ x_i | i = 1, K = (k_{ij}), i, j = 1 \ldots, N \} \)

In this case, we can make kernel predictions from predictions in \( \mathcal{H} \) (without needing pre-images)

\[
g(x, x') = \langle h_\phi(x), h_\phi(x') \rangle.
\]
Outline

1. Motivation
2. Supervised learning in kernelized output spaces
   - Output Kernel Tree
3. Ensemble methods
   - Parallel ensemble methods
   - Gradient boosting
4. Experiments
   - Image reconstruction
   - Completion of biological networks
   - Boosting
5. Conclusion and future works
A learning algorithm that solves the regression problem \((\mathcal{Y} = IR\) and \(\ell(y_1, y_2) = (y_1 - y_2)^2\)) 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).
Focus on regression trees on multiple outputs

\( Y = IR^n \) and \( \ell(y_1, y_2) = \|y_1 - y_2\|^2 \)

The algorithm is the same but:

- The best split is the one that maximizes the variance reduction:

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

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 \( S \):

\[
\text{var}\{y|S\} = \frac{1}{N} \sum_{i=1}^{N} ||y_i - \bar{y}||^2 \text{ with } \bar{y} = \frac{1}{N} \sum_{i=1}^{N} y_i
\]
Regression trees in output feature space

Let us suppose we have access to an output Gram matrix $k(y_i, y_j)$ with $k$ a kernel defined on $\mathcal{Y} \times \mathcal{Y}$ (with corresponding feature map $\phi: \mathcal{Y} \rightarrow \mathcal{F}$ such that $k(y_i, y_j) = \langle \phi(y_i), \phi(y_j) \rangle$)

The idea is to grow a multiple output regression tree in the output feature space:

- The variance becomes:

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

- Predictions at leaf nodes become pre-images of the centers of mass

$$\hat{y}_L = \phi^{-1}\left(\frac{1}{N_L} \sum_{i=1}^{N_L} \phi(y_i)\right)$$

- We need to express everything in terms of kernel values only and return to the original output space $\mathcal{Y}$
Kernelization

The variance may be written:

\[
\text{var}\{\phi(y) \mid S\} = \frac{1}{N} \sum_{i=1}^{N} \|\phi(y_i) - \frac{1}{N} \sum_{i=1}^{N} \phi(y_i)\|^2
\]

\[
= \frac{1}{N} \sum_{i=1}^{N} <\phi(y_i), \phi(y_i)> - \frac{1}{N^2} \sum_{i,j=1}^{N} <\phi(y_i), \phi(y_j)>,
\]

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:

\[
\text{var}\{\phi(y) \mid S\} = \frac{1}{N} \sum_{i=1}^{N} k(y_i, y_i) - \frac{1}{N^2} \sum_{i,j=1}^{N} k(y_i, y_j)
\]

From kernel values only, we can thus grow a regression tree that minimizes output feature space variance.
Prediction in the original output space

- Each leaf is associated with a subset of outputs from the learning sample

\[ \hat{y}_L = \phi^{-1} \left( \frac{1}{N_L} \sum_{i=1}^{N_L} \phi(y_i) \right) \]

- Generic proposal for the pre-image problem: Find the output in the leaf closest to the center of mass:

\[ \hat{y}_L = \arg \min_{y' \in \{y_1, \ldots, y_{N_L}\}} \| \phi(y') - \frac{1}{N_L} \sum_{i=1}^{N_L} \phi(y_i) \|^2 = \arg \min_{y' \in \{y_1, \ldots, y_{N_L}\}} k(y', y') - \frac{2}{N_L} \sum_{i=1}^{N_L} k(y_i, y') \]
Ensemble methods

- Parallel ensemble methods based on randomization:
  - Grow several models in parallel and average their predictions
  - Greatly improve accuracy of single regressors by reducing their variance
  - Usually, they can be applied directly (e.g., bagging, random forests, extra-trees)

- Boosting algorithms:
  - Grow the models in sequence by focusing on “difficult” examples
  - Need to be extended to regressors with kernelized outputs
  - We propose a kernelization of gradient boosting approaches (Friedman, 2001).
Parallel ensemble methods

- To make a prediction, we need to compute:

\[
\hat{y}_T(x) = \phi^{-1}\left(\frac{1}{M} \sum_{m=1}^{M} h^\phi(x; a_m)\right).
\]

- With output kernel trees as base regressors, ensemble predictions in the output feature space may be written:

\[
\frac{1}{M} \sum_{m=1}^{M} h^\phi(x; a_m) = \sum_{i=1}^{N_{LS}} k_T(x_i, x)\phi(y_i), \text{ with } k_T(x, x') = M^{-1} \sum_{m=1}^{M} k_{tm}(x, x'),
\]

where \( k_{tm}(x, x') = N_L^{-1} \) if \( x \) and \( x' \) reach the same leaf \( L \) in the \( m \)th tree, \( t_m \), and 0 otherwise.

- Predictions can then be computed by:

\[
\hat{y}_T(x) = \arg\min_{y' \in LS \mid k_T(x', x) \neq 0} k(y', y') - 2 \sum_{i=1}^{N_{LS}} k_T(x_i, x)k(y_i, y').
\]
Generic gradient boosting

General supervised learning problem:

From a learning sample \( \{(x_i, y_i)| i = 1, \ldots, N\} \) with \( x_i \in \mathcal{X} \) and \( y_i \in \mathcal{Y} \), find a function \( F : \mathcal{X} \to \mathcal{Y} \) that minimizes the expectation of some loss function \( \ell \) over the joint distribution of input/output pairs:

\[
E_{x,y}\{\ell(F(x), y)\}
\]

Boosting tries to find an approximation \( F(x) \) of the form:

\[
F(x) = F_0(x) + \sum_{m=1}^{M} \beta_m h(x; a_m),
\]

where \( h(x; a) \) is a simple parametrized function of the input variables \( x \), characterized by a vector of parameters \( a \).
“Greedy-stagewise” approach: From some starting function $F_0(x)$, for $m = 1, 2, \ldots, M$:

$$(\beta_m, a_m) = \arg \min_{\beta, a} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(x_i) + \beta h(x_i; a))$$

$$F_m(x) = F_{m-1}(x) + \beta_m h(x; a_m)$$

$\min_{\beta, a}$ may be difficult to compute $\Rightarrow$ find the function that is the closest to the steepest-descent direction in the N-dimensional data space at $F_{m-1}(x)$:

$$-g_m(x_i) = -\left[ \frac{\delta \ell(y_i, F(x_i))}{\delta F(x_i)} \right]_{F(x) = F_{m-1}(x)}$$

To generalize, find the function $h(x; a_m)$ that produces $\{h(x_i; a_m)\}_{1}^{N}$ most parallel to $-g_m$, e.g. obtained from:

$$a_m = \arg \min_{\beta, a} \sum_{i=1}^{N} (-g_m(x_i) - \beta h(x_i; a))^2.$$
Gradient Boost

1. \( F_0(x) = \arg \min_{\rho} \sum_{i=1}^{N} \ell(y_i, \rho) \)

2. For \( m = 1 \) to \( M \) do:
   1. \( y_i^m = -\left(\frac{\delta \ell(y_i, F(x_i))}{\delta F(x_i)}\right) F(x) = F_{m-1}(x), \ i = 1, \ldots, N \)
   2. \( a_m = \arg \min_{a, \beta} \sum_{i=1}^{N} (y_i^m - \beta h(x_i; a))^2 \)
   3. \( \rho_m = \arg \min_{\rho} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(x_i) + \rho h(x_i; a_m)) \)
   4. \( F_m(x) = F_{m-1}(x) + \rho_m h(x; a_m) \)

- Replace a minimization over any (differentiable) loss \( \ell \) by a least-squares function minimization (2.2) and only a single parameter optimization based on \( \ell \) (2.3)

- Can take benefit of any \( h(x; a) \) for which a feasible least-squares algorithm exists
Gradient boosting with square loss

- 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. $y_i^m = y_i - F_{m-1}(x_i), \ i = 1, \ldots, N$
   2. $a_m = \arg \min_a \sum_{i=1}^{N} (y_i^m - h(x_i; a))^2$
   3. $F_m(x) = F_{m-1}(x) + h(x; a_m)$

- e.g., $h(x; a)$ are small regression trees (Friedman’s Multiple Additive Regression Trees, MART).
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. $y_i^m = y_i - F_{m-1}(x_i)$, $i = 1, \ldots, N$
   2. $a_m = \arg \min_a \sum_{i=1}^{N} (y_i^m - h(x_i; a))^2$
   3. $F_m(x) = F_{m-1}(x) + \mu h(x; a_m)$

- e.g., $h(x; a)$ are small regression trees (Friedman’s Multiple Additive Regression Trees, MART).
- In practice, it is very useful to regularize ($\mu << 1$)
**Kernelizing the output (1/2)**

---

**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. \( \phi_i^m = \phi(y_i) - F_{m-1}^\phi(x_i), \quad i = 1, \ldots, N \)
   2. \( a_m = \text{arg min}_a \sum_{i=1}^{N} ||\phi_i^m - h^\phi(x_i; a)||^2 \)
   3. \( F_m^\phi(x) = F_{m-1}^\phi(x) + h^\phi(x; a_m) \)

- 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)
- \( F^\phi \) and \( h^\phi \) are now functions from \( \mathcal{X} \) to \( \mathcal{H} \)
### 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. \[ \phi_i^m = \phi(y_i) - F_{m-1}^\phi(x_i), \quad i = 1, \ldots, N \]
   2. \( a_m = \arg \min_a \sum_{i=1}^{N} \| \phi_i^m - h^\phi(x_i; a) \|^2 \)
   3. \[ F_m^\phi(x) = F_{m-1}^\phi(x) + h^\phi(x; a_m) \]

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

\[
h^\phi(x; a_m) = \sum_{i=1}^{N} w_i(x; a_m) \phi_i^m
\]
Kernelizing the output: learning stage

\[ K_{i,j}^m \triangleq \langle \phi_i^m, \phi_j^m \rangle = \langle \phi(y_i) - F_{m-1}^{\phi}(x_i), \phi(y_j) - F_{m-1}^{\phi}(x_j) \rangle \]
\[ = \langle \phi(y_i) - F_{m-2}^{\phi}(x_i) - h^{\phi}(x_i; a_{m-1}), \phi(y_j) - F_{m-2}^{\phi}(x_j) - h^{\phi}(x_j; a_{m-1}) \rangle \]
\[ = \langle \phi_i^{m-1}, \phi_j^{m-1} \rangle - \langle \phi_i^{m-1}, h^{\phi}(x_j; a_{m-1}) \rangle - \langle h^{\phi}(x_i; a_{m-1}), \phi_j^{m-1} \rangle \]
\[ + \langle h^{\phi}(x_i; a_{m-1}), h^{\phi}(x_j; a_{m-1}) \rangle. \]

Using \( h^{\phi}(x; a_{m-1}) = \sum_{i=1}^{N} w_i(x; a_{m-1}) \phi_i^{m-1} \) and \( K_{i,j}^{m-1} \triangleq \langle \phi_i^{m-1}, \phi_j^{m-1} \rangle \):

\[ K_{i,j}^m = K_{i,j}^{m-1} - \sum_{l=1}^{N} w_l(x_j; a_m) K_{i,l}^{m-1} - \sum_{l=1}^{N} w_l(x_i; a_m) K_{i,l}^{m-1} \]
\[ + \sum_{k,l=1}^{N} w_k(x_i; a_m) w_l(x_j; a_m) K_{k,l}^{m-1}, \]
Output kernel based boosting: learning

Input: a learning sample \( \{(x_i, y_i)\}_{i=1}^{N} \) and an output Gram matrix \( K \) (with \( K_{i,j} = k(y_i, y_j) \)).

Output: an ensemble of weight functions \( \{(w_i(x; a_m))_{i=1}^{N}\}_{m=0}^{M} \).

1. \( w_i(x; a_0) \equiv 1/N, \ W_{i,j}^0 = 1/N, \forall i, j = 1, \ldots, N, \ K^0 = K. \)

2. For \( m = 1 \) to \( M \) do:
   1. \( K^m = (I - W^{m-1})' K^{m-1} (I - W^{m-1}). \)
   2. Apply the base learner to the output Gram matrix \( K^m \) to get a model \( (w_i(x; a_m))_{i=1}^{N} \).
   3. Compute \( W_{i,j}^m = w_i(x_j; a_m) \), \( i, j = 1, \ldots, N \) from the resulting model.

\( (K^1 \) is the original output Gram matrix centered)
In the output feature space, predictions are of the form:

\[ F_M^\phi(x) = \sum_{i=1}^{N} w_i^F(x) \phi(y_i). \]

Output and kernel predictions are then obtained from:

\[ F(x) = \arg \min_{y' \in Y} \| \phi(y') - \sum_{i=1}^{N} w_i^F(x) \phi(y_i) \|^2 \]

\[ = \arg \min_{y' \in Y} k(y', y') - 2 \sum_{i=1}^{N} w_i^F(x) k(y_i, y'). \]

\[ \hat{k}(x_1, x_2) = \sum_{i=1}^{N} \sum_{j=1}^{N} w_i^F(x_1) w_j^F(x_2) K_{i,j}. \]

A recursive algorithm may be devised to compute the weight vector

\[ w^F(x) = (w_1^F(x), \ldots, w_N^F(x)). \]
For the $m$th model, we have:

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

where each output feature vector $\phi_i^m$ may be further written:

$$\phi_i^m = \sum_{j=1}^{N} O_{i,j}^m \phi(y_j)$$

The following recursion computes the $N \times N$ matrices $O^m$:

- $O^0 = I$
- $O^m = O^{m-1} - W^{m-1} O^{m-1}$, $\forall m = 1, \ldots, M$

Denoting by $p^m(x)$ the (line) vector $(w_1(x; a_m), \ldots, w_N(x; a_m))$, we thus have:

$$w^F(x) = \sum_{m=0}^{M} p^m(x) O^m$$
Output kernel based boosting: predictions

**Input:** a test sample of $Q$ input vectors, $\{x'_1, \ldots, x'_Q\}$.

**Output:** a prediction $F_M^Y(x'_i) \in \mathcal{Y}$ for each input $x'_i$, $i = 1, \ldots, Q$ and an output kernel matrix prediction $\hat{K}$ with $\hat{K}_{i,j} = \langle F_M^\phi(x'_i), F_M^\phi(x'_j) \rangle$, $i, j = 1, \ldots, Q$.

1. $O^0 = I$, $W^{0}_{i,j} = 1/N$, $\forall i, j = 1, \ldots, N$, $W^F_{i,j} = \frac{1}{N}$, $\forall i = 1, \ldots, Q$, $j = 1, \ldots, N$
2. For $m = 1$ to $M$ do:
   1. $O^m = O^{m-1} - W^{m-1}^T O^{m-1}$.
   2. Compute the $Q \times N$ matrix $P^m$ with $P^m_{i,j} = w_j(x'_i; a_m)$, $\forall i = 1, \ldots, Q$, $\forall j = 1, \ldots, N$.
   3. Set $W^F$ to $W^F + P^m O^m$.
   4. Compute $W^m_{i,j} = w_i(x'_j; a_m)$, $\forall i, j = 1, \ldots, N$ from the $m$th model.
## Output kernel based boosting: predictions

**Input:** a test sample of $Q$ input vectors, $\{x'_1, \ldots, x'_Q\}$.

**Output:** a prediction $F^\gamma_M(x'_i) \in \mathcal{Y}$ for each input $x'_i$, $i = 1, \ldots, Q$ and an output kernel matrix prediction $\hat{K}$ with $\hat{K}_{i,j} = \langle F^\phi_M(x'_i), F^\phi_M(x'_j) \rangle$, $i, j = 1, \ldots, Q$.

1. $O^0 = I$, $W^0_{i,j} = 1/N$, $\forall i, j = 1, \ldots, N$, $W^F_{i,j} = \frac{1}{N}$, $\forall i = 1, \ldots, Q$, $j = 1, \ldots, N$

2. For $m = 1$ to $M$ do:
   1. $O^m = O^{m-1} - W^{m-1}O^{m-1}$.
   2. Compute the $Q \times N$ matrix $P^m$ with $P^m_{i,j} = w_j(x'_i; a_m)$, $\forall i = 1, \ldots, Q$, $\forall j = 1, \ldots, N$.
   3. Set $W^F$ to $W^F + P^mO^m$.
   4. Compute $W^m_{i,j} = w_i(x'_j; a_m)$, $\forall i, j = 1, \ldots, N$ from the $m$th model.

3. To compute predictions in the output space:
   1. Compute $S = 1_{Q \times 1} \text{diag}(K)' - 2W^F K$.
   2. $F^\gamma_M(x'_i) = y_k$ with $k = \arg \min_{j=1,\ldots,N} S_{i,j}$, $\forall i = 1, \ldots, Q$. 
Output kernel based boosting: predictions

**Input:** a test sample of $Q$ input vectors, $\{x'_1, \ldots, x'_Q\}$.

**Output:** a prediction $F^Y_M(x'_i) \in \mathcal{Y}$ for each input $x'_i$, $i = 1, \ldots, Q$ and an output kernel matrix prediction $\hat{K}$ with $\hat{K}_{i,j} = \langle F^\phi_M(x'_i), F^\phi_M(x'_j) \rangle$, $i, j = 1, \ldots, Q$.

1. $O^0 = I$, $W^0_{i,j} = 1/N$, $\forall i,j = 1, \ldots, N$, $W^F_{i,j} = \frac{1}{N}$, $\forall i = 1, \ldots, Q, j = 1, \ldots, N$

2. For $m = 1$ to $M$ do:
   1. $O^m = O^{m-1} - W^{m-1}O^{m-1}$.
   2. Compute the $Q \times N$ matrix $P^m$ with $P^m_{i,j} = w_j(x'_i; a_m)$, $\forall i = 1, \ldots, Q$, $\forall j = 1, \ldots, N$.
   3. Set $W^F$ to $W^F + P^mO^m$.
   4. Compute $W^m_{i,j} = w_i(x'_j; a_m)$, $\forall i,j = 1, \ldots, N$ from the $m$th model.

3. To compute predictions in the output space:
   1. Compute $S = I_{Q \times 1} \text{diag}(K)' - 2W^FK$.
   2. $F^Y_M(x'_i) = y_k$ with $k = \arg \min_{j=1,\ldots,N} S_{i,j}$, $\forall i = 1, \ldots, Q$.

4. To compute kernel predictions:
   1. $\hat{K} = W^FKW^F$.
With OK3 as base learner

- The prediction of the $m$th tree at some point $x$ is given by:

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

with $w_i(x; a_m) = \frac{1}{N_L}$ if $x$ and $x_i$ reach the same leaf of size $N_L$, 0 otherwise.

- The matrices $W^m$ are symmetric and positive definite.
  - $\phi(x)$ is an $N$-dimensional vector whose $i$-th component is equal to $1/\sqrt{N_L}$ when $x$ reaches the leaf of size $N_L$ that contains $x_i$, 0 otherwise.

- To constrain the tree complexity, we fix the number of splits to a small number $J$ (using a best first strategy to grow the tree).

- When $\mathcal{Y} = IR$ and $k(y_1, y_2) = y_1 y_2$, we get MART.
Direct features from tree based methods

- **Interpretability:**
  - Single tree: a single tree provides a rule-based model that is directly interpretable
  - Ensemble of trees: a ranking of the features according to their relevance can be obtained by summing the total variance reduction over all nodes where the feature appears and normalizing over all variables

- **Computational efficiency:**
  - Learning stage: node splitting goes from $O(N)$ for standard trees to $O(N^2)$ for OK3. Matrix updates for gradient boosting are also quadratic in $N$ with output kernel trees.
  - Prediction stage: pre-image computation is $O(N_L^2)$, where $N_L$ is the size of the leaf for a single tree and the support of $k_T(x, \cdot)$ for an ensemble.
Motivation

2 Supervised learning in kernelized output spaces

3 Output Kernel Tree

4 Ensemble methods
   - Parallel ensemble methods
   - Gradient boosting

5 Experiments
   - Image reconstruction
   - Completion of biological networks
   - Boosting

6 Conclusion and future works
First application: image reconstruction (Weston et al., NIPS 2002)

\{ (\begin{array}{c} 3 \\ 6 \\ 8 \end{array}), (\begin{array}{c} 9 \\ 2 \\ 1 \end{array}), ... , (\begin{array}{c} 0 \\ 1 \\ 7 \end{array}) \} \)

- Predict the bottom half of an image representing a handwritten digit from its top half
- Subset of the USPS dataset: 1000 images, 16×16 pixels
  - Input variables: 8×16(=128) continuous variables
  - Output kernel: radial basis-function (RBF) kernel:
    \[ k(y, y') = \exp\left( -\frac{\|y - y'\|^2}{2\sigma^2} \right) \]
- Protocol
  - Estimation of the average RBF loss by 5-fold CV
  - Comparison with k-NN and Kernel Dependency Estimation (KDE, a full kernel-based method for structured output prediction fitting a ridge regression model on each direction found by kernel PCA)
Illustration: accuracy results

<table>
<thead>
<tr>
<th>Method</th>
<th>RBF error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1.0853</td>
</tr>
<tr>
<td>Best achievable</td>
<td>0.3584</td>
</tr>
<tr>
<td>k-NN</td>
<td>0.7501</td>
</tr>
<tr>
<td>KDE linear</td>
<td>0.7990</td>
</tr>
<tr>
<td>KDE RBF</td>
<td>0.6778</td>
</tr>
<tr>
<td>OK3+Single trees</td>
<td>0.9013</td>
</tr>
<tr>
<td>OK3+Bagging</td>
<td>0.7337</td>
</tr>
<tr>
<td>OK3+Extra-trees</td>
<td>0.6949</td>
</tr>
</tbody>
</table>

Examples of predictions

![Examples of predictions](image)

Feature ranking

![Feature ranking](image)
From a known network where each vertex is described by some input feature vector $x$, predict the edges involving new vertices described by their input feature vector.
A general solution based on a kernelized output space

- Define a kernel $k$ on pairs of vertices such that $k(v, v')$ encodes the proximity of vertices in the graph.
- Use a machine learning method that can handle a kernelized output space to get an approximation $g(x(v), x(v'))$ of the kernel value between $v$ and $v'$ described by their input feature vectors $x(v)$ and $x(v')$
- Connect these two vertices if $g(x(v), x(v')) > k_{th}$

(by varying $k_{th}$ we get different tradeoffs between true positive and false positive rates)
A kernel on graph nodes

- Diffusion kernel (Kondor and Lafferty, 2002):
  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:

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

- As the diffusion coefficient $\beta$ increases, kernel values diffuse more completely through the graph.
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 of different tree based methods

Diffusion kernel as a graph kernel, 10-fold cross-validation, ensembles of 100 output kernel trees
Comparison of different sets of features

Diffusion kernel as a graph kernel, 10-fold cross-validation, ensembles of 100 output kernel trees, extra-trees randomization method
Comparison with full kernel based methods

|--------|--------|-----| |--------|--------|-----|
| expr   | 0.851  | 0.776 | | expr   | 0.714  | 0.706 |
| phy    | 0.693  | 0.767 | | phy    | 0.815  | 0.747 |
| loc    | 0.725  | 0.788 | | loc    | 0.587  | 0.577 |
| y2h    | 0.790  | 0.612 | | **All**| **0.847**| **0.804** |
| **All**| **0.910**| **0.939**| |                |        |    |


[2] Yamanishi et al., ISMB 2005: compare a kernel canonical correlation analysis based solution and a metric learning approach
Evolution of the AUC when $x\%$ of the edges are randomly deleted in the learning sample (OK3+ET, 100 trees)

<table>
<thead>
<tr>
<th></th>
<th>0%</th>
<th>20%</th>
<th>50%</th>
<th>80%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protein network</td>
<td>0.910</td>
<td>0.906</td>
<td>0.896</td>
<td>0.883</td>
</tr>
<tr>
<td>Enzyme network</td>
<td>0.844</td>
<td>0.800</td>
<td>0.812</td>
<td>0.753</td>
</tr>
</tbody>
</table>
Interpretability: rules and clusters (an example with a protein-protein network)
Interpretability: feature ranking

<table>
<thead>
<tr>
<th>#</th>
<th>Att.</th>
<th>Imp</th>
<th>#</th>
<th>Att.</th>
<th>Imp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>loc - nucleolus</td>
<td>0.021</td>
<td>1</td>
<td>phy - dre</td>
<td>0.011</td>
</tr>
<tr>
<td>2</td>
<td>expr (Spell.) - elu 120</td>
<td>0.013</td>
<td>2</td>
<td>phy - rno</td>
<td>0.009</td>
</tr>
<tr>
<td>3</td>
<td>loc - cytoplasm</td>
<td>0.012</td>
<td>3</td>
<td>expr (Eisen) - cdc15 120m</td>
<td>0.008</td>
</tr>
<tr>
<td>4</td>
<td>expr (Eisen) - sporulation ndt80 early</td>
<td>0.012</td>
<td>4</td>
<td>phy - ecu</td>
<td>0.008</td>
</tr>
<tr>
<td>5</td>
<td>loc - nucleus</td>
<td>0.012</td>
<td>5</td>
<td>expr (Eisen) - cdc15 160m</td>
<td>0.008</td>
</tr>
<tr>
<td>6</td>
<td>expr (Eisen) - sporulation 30m</td>
<td>0.011</td>
<td>6</td>
<td>phy - pfa</td>
<td>0.007</td>
</tr>
<tr>
<td>7</td>
<td>expr (Eisen) - sporulation ndt80 middle</td>
<td>0.010</td>
<td>7</td>
<td>phy - mmu</td>
<td>0.007</td>
</tr>
<tr>
<td>8</td>
<td>expr (Spell.) - alpha 14</td>
<td>0.010</td>
<td>8</td>
<td>loc - cytoplasm</td>
<td>0.006</td>
</tr>
<tr>
<td>9</td>
<td>expr (Spell.) - elu 150</td>
<td>0.010</td>
<td>9</td>
<td>expr (Eisen) - cdc15 30m</td>
<td>0.005</td>
</tr>
<tr>
<td>10</td>
<td>loc - mitochondrion</td>
<td>0.009</td>
<td>10</td>
<td>expr (Eisen) - elutriation 5.5hrs</td>
<td>0.005</td>
</tr>
</tbody>
</table>

Protein-protein interactions

Enzyme network

Kernelized output spaces

Experiments

25 janvier 2007
Experiments with boosting

On the image completion task (#LS=200,#TS=800):

- $\nu=0.5$, $J=10$
- $\nu=0.05$, $J=10$

On the network completion problem (#LS=334,#TS=334):

- $\nu=0.25$, $J=10$
- $\nu=0.05$, $J=10$

($Err_\phi$ is the error of $F^\phi_M(x)$ (i.e. in $\mathcal{H}$), $Err^Y$ is the error of the pre-image)
### Experiments with boosting

<table>
<thead>
<tr>
<th>Method</th>
<th>$Err^Y(TS)$</th>
<th>Method</th>
<th>AUC(TS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK3 (single trees)</td>
<td>1.0399</td>
<td>OK3 (single trees)</td>
<td>0.6001</td>
</tr>
<tr>
<td>OK3+Bagging</td>
<td>0.8643</td>
<td>OK3+Bagging</td>
<td>0.7100</td>
</tr>
<tr>
<td>OK3+ET</td>
<td>0.8169</td>
<td>OK3+ET</td>
<td>0.7884</td>
</tr>
<tr>
<td>OK3+OKBoost</td>
<td>0.8318</td>
<td>OK3+OKBoost</td>
<td>0.7033</td>
</tr>
<tr>
<td>OK3+OKBoost+ET</td>
<td>0.8071</td>
<td>OK3+OKBoost+ET</td>
<td>0.7811</td>
</tr>
</tbody>
</table>

$(\mu = 0.01, M = 500, \text{Tree size } J \text{ determined by 5-fold CV})$
A new method for prediction in kernelized output spaces
- When used in a single tree, it can provide interpretable results in the form of a rule based model.
- When used in an ensemble of trees, it provides competitive accuracy and can rank the input features according to their relevance

Future works:
- Other frameworks: transduction (straightforward), semi-supervised...
- Other applications: hierarchical classification, sequence predictions,...
- Analysis of the role played by the output kernel in the cost function (regularization, output kernel learning)
- Improving gradient boosting by explicit regularization and use of other base learners
- Study links between this approach and Taskar’s approaches (features on input/outputs)
Another base learner for boosting (1/2)

- A simpler base learner:
  1. Find a direction \( \mathbf{v} = \sum_{i=1}^{N} v_i \phi(y_i^m), \| \mathbf{v} \| = 1 \), in the output feature space
  2. Project the data in this direction to obtain a training set:
     \[
     \{(x_i, \langle \phi(y_i^m), \mathbf{v} \rangle) | i = 1, \ldots, N\}
     \]
     and use any regression method to find an approximation \( f_m(x_i) \) of \( \langle \phi(y_i^m), \mathbf{v} \rangle \)
  3. Output the function:
     \[
     h_\phi(x; a_m) = f_m(x) \mathbf{v} = \sum_{i=1}^{N} f_m(x) v_i \phi(y_i^m)
     \]
How to select the direction \( v \)?

- Choose the direction of maximum variance (kernel PCA)
- Choose a direction at random
- For computational efficiency reason, choose a direction among the outputs in the learning sample.

Regression method:

- Any regression method can be plugged
- With regression trees, we get MART when \( \mathcal{Y} = IR \) and \( k(y_1, y_2) = y_1 y_2 \)