Regression tree ensembles in the perspective of kernel-based methods

Louis Wehenkel

Systems and Modeling, Department of EE & CS
Systems Biology and Chemical Biology, GIGA-Research
University of Liège, Belgium

MPI for Biological Cybernetics - Tübingen - October 2009

http://www.montefiore.ulg.ac.be/∼lwh/

Motivation/Overview

A. Highlight some possible connections and combinations of tree-based and kernel-based methods.

B. Present in this light some extensions of tree-based methods to learn with non standard data.

Motivation/Overview

A. Highlight some possible connections and combinations of tree-based and kernel-based methods.

B. Present in this light some extensions of tree-based methods to learn with non standard data.

Part I: Standard view of tree-based regression

- Standard regression tree induction
- Ensembles of extremely randomized trees

Part II: Kernel view of tree-based regression

- Input space kernel formulation of tree-based models
- Supervised learning in kernelized output spaces
- Semi-supervised learning and handling censored data

Part III: Handling structured input spaces with tree-based methods

- Content-based image retrieval
- Image classification, segmentation
- Other structured input spaces

Part I

Standard view of tree-based regression

Typical batch-mode supervised regression (Reminder)

- From an iid sample $ls \sim (P(x, y))^N$ (inputs, outputs)
  extract a model $\hat{y}_{ls}(x)$ to predict outputs (regression tree, MLP, ...)

Typical batch-mode supervised regression (Reminder)

- From an iid sample $ls \sim (P(x, y))^N$ (inputs, outputs)
  extract a model $\hat{y}_{ls}(x)$ to predict outputs (regression tree, MLP, ...)

$$ls = \begin{array}{cc|c} x_1 & x_2 & y \\ \hline 0.4 & 0.4 & 0.2 \\ 0.4 & 0.8 & 0.0 \\ 0.6 & 0.6 & 0.5 \\ 0.8 & 0.6 & 0.8 \\ 0.8 & 0.8 & 1.0 \end{array} \rightarrow \boxed{\text{BMSL}} \rightarrow \hat{y}_{\text{lin}}(x) = \alpha_{ls} x_1 + \beta_{ls} x_2 + \gamma_{ls}$$

Inputs are often high dimensional: e.g. $x \in \mathbb{R}^n$, $n \gg 100$
Outputs are typically simpler: e.g. for regression $y \in \mathbb{R}$

Typical batch-mode supervised regression (Reminder)

- From an iid sample $ls \sim (P(x, y))^N$ (inputs, outputs)
  extract a model $\hat{y}_{ls}(x)$ to predict outputs (regression tree, MLP, ...)

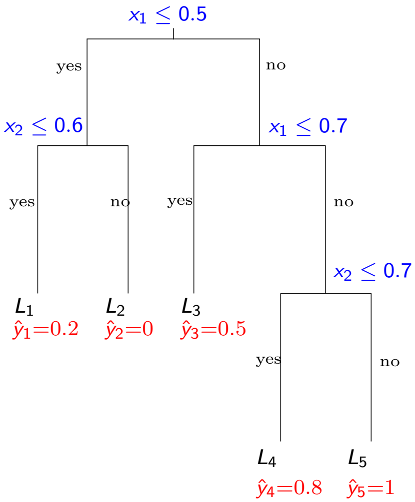
$ls =$

x_1	x_2	y
0.4	0.4	0.2
0.4	0.8	0.0
0.6	0.6	0.5
0.8	0.6	0.8
0.8	0.8	1.0

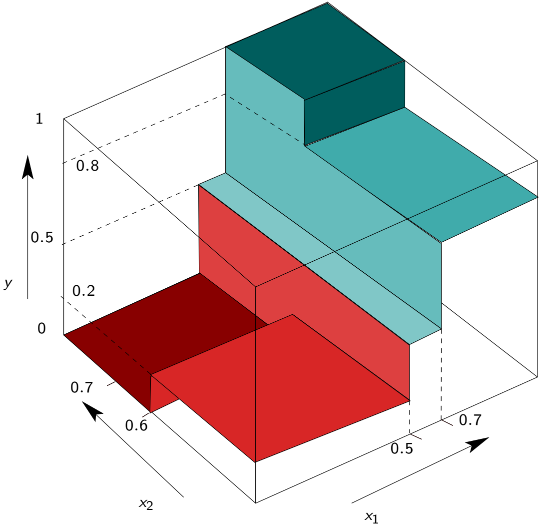
$$\rightarrow \boxed{\text{BMSL}} \rightarrow \hat{y}_{\text{lin}}(x) = \alpha_{ls}x_1 + \beta_{ls}x_2 + \gamma_{ls}$$

  Inputs are often high dimensional: e.g. $x \in \mathbb{R}^n$, $n \gg 100$
  Outputs are typically simpler: e.g. for regression $y \in \mathbb{R}$
- Typical objectives of BMSL algorithm design:
  - accuracy of predictions, measured by a loss function $\ell(y, \hat{y})$
  - interpretability, computational scalability (wrt N and/or n)

A regression tree approximator $\hat{y}_t(x_1, x_2)$



Geometrical representation of $\hat{y}_t(x_1, x_2)$



Single regression trees: learning algorithm

A. Top-down tree growing by greedy recursive partitioning

⇒ Reduce empirical loss as quickly as possible:

- Start with complete *ls*
- For an input variable $x_i \in \mathbb{R}$ find optimal threshold to split
- Split according to best (input variable,threshold) combination
- Carry on with the resulting subsets
- ... until empirical loss has been sufficiently reduced

B. Bottom-up tree pruning

⇒ Reduce overfitting to learning sample

- Generate sequence of shrinking trees
- Evaluate their accuracy on independent sample (or by CV)
- Select best tree in sequence

Node splitting and leaf labeling (when using quadratic loss ℓ)

- The best label for a leaf L is the locally optimal constant $\hat{y}_L$ in terms of quadratic loss:

$$\hat{y}_L = \arg\min_{y \in \mathbb{R}} \sum_{i \in s(L)} (y^i - y)^2 = \frac{1}{\#s(L)} \sum_{i \in s(L)} y^i$$

where $s(L)$ denotes the sub-sample reaching L, and $\#s(L)$ its cardinality.

- The best split locally maximizes the quadratic loss reduction:

$$\text{Score}_R(split, s) = (\#s)\text{var}\{y|s\} - (\#s_l)\text{var}\{y|s_l\} - (\#s_r)\text{var}\{y|s_r\},$$

where $\text{var}\{y|s\}$ denotes the empirical variance of y computed from s:

$$(\#s)\text{var}\{y|s\} = \sum_{i \in s}(y^i - \frac{1}{\#s}\sum_{i \in s} y^i)^2 = \min_{y \in \mathbb{R}} \sum_{i \in s}(y^i - y)^2.$$

Illustration: tree growing=greedy empirical loss reduction

(We show in blue the input variable that is used to split at each node)

x_1	x_2	y
0.4	0.4	0.2
0.4	0.8	0.0
0.6	0.6	0.5
0.8	0.6	0.8
0.8	0.8	1.0

$\longrightarrow$

x_1	x_2	y
0.4	0.4	0.2
0.4	0.8	0.0

$var\{y\}= 0.01$

$\longrightarrow$ $\hat{y} = 0.2$; $\hat{y} = 0.0$

x_1	x_2	y
0.6	0.6	0.5
0.8	0.6	0.8
0.8	0.8	1.0

$\longrightarrow$ $\hat{y} = 0.5$

x_1	x_2	$\hat{y}$
0.8	0.6	0.8
0.8	0.8	1.0

$\longrightarrow$ $\hat{y} = 0.8$; $\hat{y} = 1.0$

$\text{var}\{y|ls\}=0.136$ $\quad$ $\text{var}\{y\}=0.042$ $\quad$ $\text{var}\{y\}=0.01$ $\quad$ $\text{var}\{y\}=0$

Determination of the importance of input variables

- For each input variable and each test node where it is used
  - multiply score by relative subset size $(N(node)/N(ls))$
  - cumulate these values
  - normalize to compute relative total variance reduction brought by each input variable
- E.g. in our illustrative example:
  - x_1: $5/5 \times 0.107$ at root node $+ 3/5 \times 0.035$ at second test node $= 0.128$
  - x_2: $2/5 \times 0.01$ at one node $+ 2/5 \times 0.01$ at other node $= 0.008$
  - x_1 brings 94% and x_2 brings 6% of variance reduction.

Standard regression trees: strengths and weaknesses

- Universal approximation/consistency
- Robustness to outliers
- Robustness to irrelevant attributes
- Invariance to scaling of inputs
- Good interpretability
- Very good computational efficiency and scalability
- Very high *training variance*
  - The trees depend a lot on the random nature of the *ls*
  - This variance increases with the depth of the tree
  - But, even for pruned trees the training variance remains high
  - As a result, accuracy of tree based models is typically low

Tree-based regression

The bagging idea for variance reduction

Observation:

- Let $\{ls_i\}_{i=1}^{M}$ be M samples of size N drawn from $P_{X,Y}$ (i.i.d.) and,
- let $A(ls_i)$ be a regression model obtained from ls_i by some algo A.
- Denote by $A_M = M^{-1} \sum_{i=1}^{M} A(ls_i)$.

Then the following holds true:

$$\boxed{\overline{MSE}\{A_M\} = MSE\{E\{y|x\}\} + bias^2\{A\} + M^{-1} variance\{A\}}$$

I.e., if the variance of A is high compared to its bias, then A_M may be significantly more accurate than A, even for small values of M

The bagging idea for variance reduction

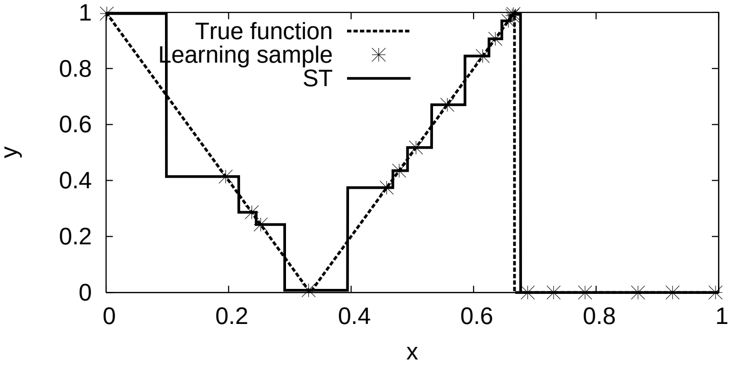
But, in practice, we have only a single learning sample of size N...

(Why should we split this *ls* into subsamples?)

The bagging trick: (Leo Breiman, mid nineties)

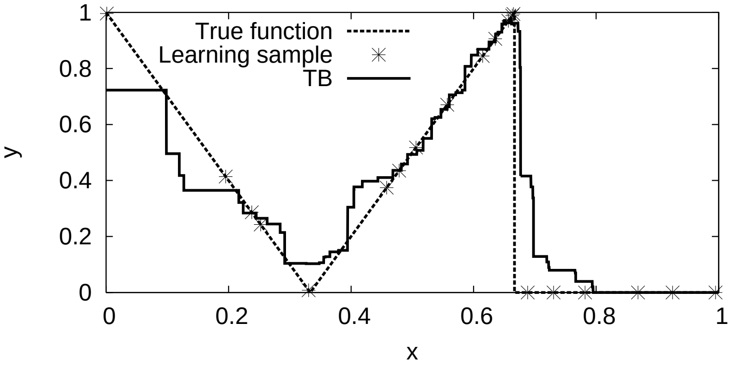
- Replace sampling from the population by sampling (with replacement) from the given *ls*
- Bagging=bootstrap+aggregating
  - $\rightarrow$ generate M bootstrap copies of ls, $\{\hat{ls}_i\}_{i=1}^{M}$
  - $\rightarrow$ build M models $A(\hat{ls}_i), i = 1 \ldots, M$
  - $\rightarrow$ construct model as average prediction $M^{-1}\sum_{i=1}^{M} A(\hat{ls}_i)$

Geometric properties (of Single Trees)



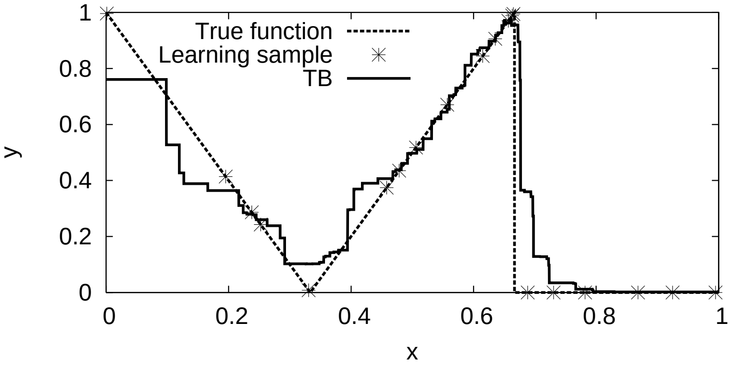
A single fully developed regression tree.

Geometric properties



Bagged trees: with $M = 100$ trees in the ensemble.

Geometric properties



Bagged trees: with $M = 1000$ trees in the ensemble.

Tree-Bagging and Perturb & Combine

Observations:

- Bagging reduces variance significantly, but not totally
- Because bagging optimizes thresholds and attribute choices on bootstrap samples, its variance reduction is limited and its computational cost is high
- Bagging increases bias, because bootstrap samples have less information than original sample (67%)

Idea:

- Since bagging works by randomization, why not randomize tree growing procedure in other ways ?
- Many other Perturb & Combine methods have been proposed along this idea to further improve bagging...

Tree-based regression

- Single regression tree induction
- Ensembles of bagged regression trees
- Ensembles of totally and extremely randomized trees

Ensembles of totally randomized trees

Basic observations:

- Variance of trees comes from the greedy split optimization
- Much of this variance is due to the choice of thresholds

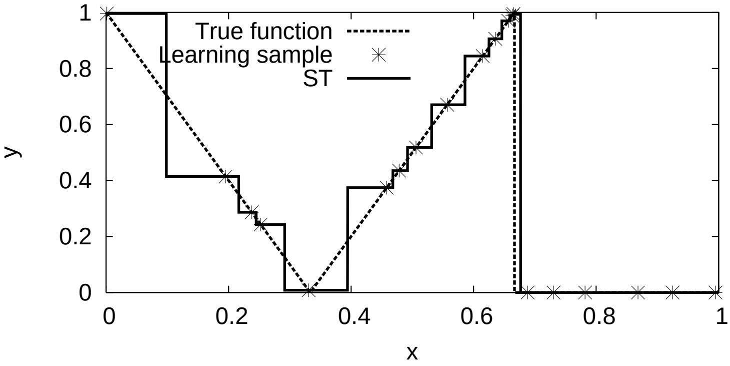
Ensembles of totally randomized trees:

- Develop nodes by selecting random attribute and threshold
- Develop tree completely (no pruning) on full *ls*
- Average the predictions of many trees (e.g. $M = 100 \ldots 1000$)

Basic properties of this method:

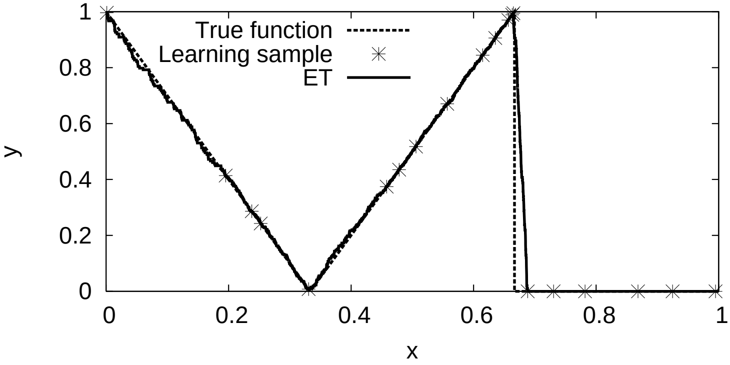
- Ultra-fast tree growing
- Tree structures are independent of outputs of *ls*
- If $M \to \infty \Rightarrow$ piece-wise (multi-)linear interpolation of *ls*

Geometric properties (of Single Trees)



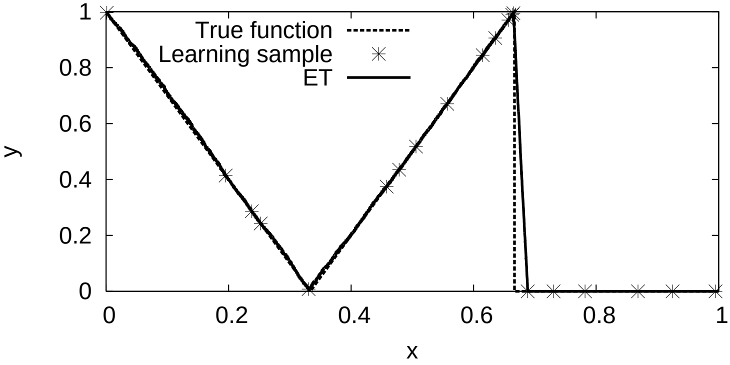
A single fully developed regression tree.

Geometric properties



Random trees: with $M = 100$ trees in the ensemble.

Geometric properties



Random trees: with $M = 1000$ trees in the ensemble.

Extremely randomized trees

Observations:

- Totally randomized trees are not robust w.r.t. irrelevant inputs
- Totally randomized trees are not robust w.r.t. noisy outputs

Solutions:

- Instead of splitting totally at random
  - Select a few (say K) inputs and thresholds at random
  - Evaluate empirical loss reduction and select the best one
- Stop splitting as soon as sample becomes too small ($n_{\min}$)

Extra-Trees algorithm has two additional parameters:

- Attribute selection strength K
- Smoothing strength $n_{\min}$

Extra-Trees: strengths and weaknesses

- Universal approximation/consistency
- Robustness to outliers
- Robustness to irrelevant attributes
- Invariance to scaling of inputs
- Loss of interpretability w.r.t. standard trees
- Very good computational efficiency and scalability
- Very low variance
- Very good accuracy

NB: straightforward generalization to discrete inputs and outputs.

NB: straightforward extensions of feature importance measure.

Further reading about Extra-Trees

P. Geurts, D. Ernst, and L. Wehenkel.
Extremely randomized trees.
Machine Learning, Volume 36, Number 1, pp. 3-42 - 2006.

P. Geurts, D. deSeny, M. Fillet, M-A. Meuwis, M. Malaise, M-P. Merville, and L. Wehenkel.
Proteomic mass spectra classification using decision tree based ensemble methods.
Bioinformatics, Vol. 21, No 14, pp 3138–3145 - 2005.

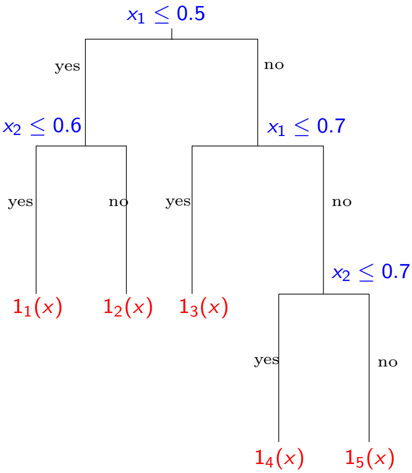
V.A. Huynh-Thu, L. Wehenkel, and P. Geurts.
Exploiting tree-based variable importances to selectively identify relevant variables.
JMLR: Workshop and Conference Proceedings, Vol. 4, pp. 60-73 - 2008.

Part II

Kernel view of regression trees

From tree structures to kernels (Step 1: exploiting a tree structure)

- A tree structure partitions $\mathcal{X}$ into ℓ regions corresponding to its leaves.
- Let $\phi(x) = (1_1(x), \ldots, 1_\ell(x))^T$ be the vector of characteristic functions of these regions: $\phi^T(x)\phi(x')$ defines a positive kernel over $\mathcal{X} \times \mathcal{X}$.
- Alternatively, we can simply say that a tree t induces the kernel $k_t(x, x')$ over the input space $\mathcal{X}$, defined by $k_t(x, x') = 1$ (or 0) if x and x' reach (or not) the same leaf of t.
- NB: $k_t(x, x')$ is a very discrete kernel: two inputs are either totally similar or totally dissimilar according to k_t.



From tree structures to kernels (Step 2: exploiting a sample $(x^i)_{i=1}^N$)

- Define the weighted feature map by $(\frac{1_1(x)}{\sqrt{N_1}}, \ldots, \frac{1_\ell(x)}{\sqrt{N_\ell}})^T$, where N_i denotes the number of samples which reach the i-th leaf.
- Denote by $k_t^{ls} : X \times X \rightarrow \mathbb{Q}$ the resulting kernel.
- Note that k_t^{ls} is less discrete than k_t: two inputs that fall together in a "small" leaf are more similar than two inputs falling together in a "big" leaf.
- With k_t^{ls}, the predictions defined by a tree t induced from a sample ($ls = ((x^1, y^1), \ldots, (x^N, y^N))$) can be computed by:

$$\hat{y}_t(x) = \sum_{i=1}^{N} y^i k_t^{ls}(x^i, x).$$

Kernel defined by an ensemble of trees

- Kernel defined by a tree ensemble $\mathcal{T} = \{t_1, t_2, \ldots, t_M\}$:

$$k^{ls}_{\mathcal{T}}(x, x') = M^{-1} \sum_{j=1}^{M} k^{ls}_{t_j}(x, x')$$

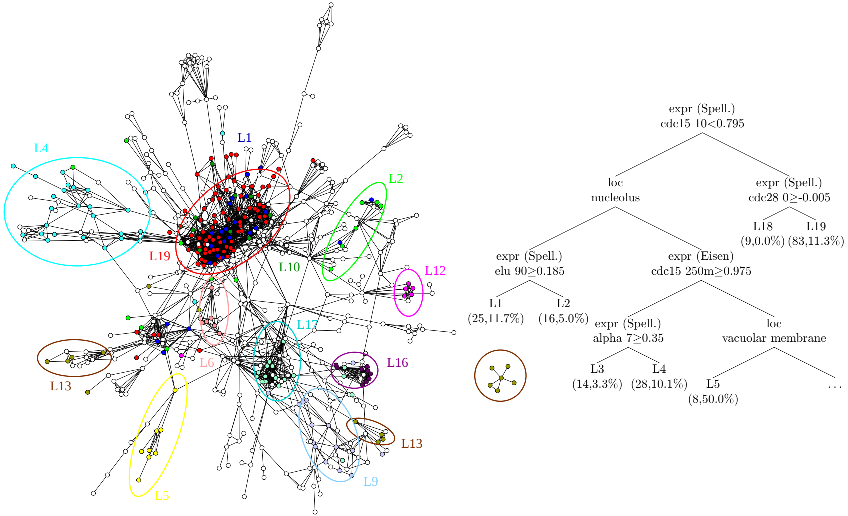
- Model defined by a tree ensemble $\mathcal{T}$:

$$\hat{y}_{\mathcal{T}}(x) = M^{-1} \sum_{j=1}^{M} \hat{y}_{t_j}(x) = M^{-1} \sum_{j=1}^{M} \sum_{i=1}^{N} y^i k^{ls}_{t_j}(x^i, x)$$

$$= \sum_{i=1}^{N} y^i M^{-1} \sum_{j=1}^{M} k^{ls}_{t_j}(x^i, x) = \sum_{i=1}^{N} y^i k^{ls}_{\mathcal{T}}(x^i, x)$$

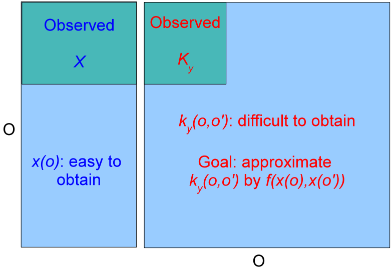
- $k^{ls}_{\mathcal{T}}(x, x')$ essentially counts the number of trees in which x and x' reach the same leaf; in this process the leaves are down-weighted by their number of learning samples.

Completing/understanding protein-protein interactions



Supervised learning of kernels: problem formulation

- Given a sample of objects $(o_i)_{i=1}^N \in \mathcal{O}^N$ of which we know the value of some kernel $k_y(o, o')$ in the form of a Gram matrix, and for which we may easily obtain some input features $x(o)$,
- we want to compute a good approximation of the target kernel k_y from input space features x.



- $X \in \mathcal{X}^N$ denotes the vector of input space descriptions over the sample.
- $k_y^{i,j}$ denotes the target kernel values in our sample and $K_y \in \mathbb{R}^{N \times N}$ the corresponding Gram matrix.

Supervised learning of kernels: applications

Supervised learning of a kernel consists of determining, from a sample of joint observations of $x(o_i)$ and $k_y(o_i, o_j)$ over some space $\mathcal{O}$, a function $f(x(o), x(o'))$ approximating well $k_y(o, o')$.

Examples

- Learning over structured output spaces, where the output space structure is defined by a kernel (both $k_y(o_i, o_j)$ and $y(o_i)$ are given for some objects) (e.g. sequences, images, graphs as outputs)
- Graph completion, where the goal is to infer a neighborhood function among objects in terms of features describing them (only a partial graph is given) (e.g. predicting protein interactions from their annotations)
- Problems where we want to compute a "proxy" of a kernel among objects which can not be determined exactly (only a subsample of $k_y(o_i, o_j)$ values is given) (e.g. measuring similarities among tertiary protein structures, as a function of their primary structure)

Rationale of the proposed approach

- Suppose that instead of K_y we dispose of "compatible" output space feature vectors Y, i.e. such that $y(o)^T y(o') = k_y(o, o')$.
- Then we could grow trees by a LMS approach over $\mathcal{Y}$, to compute $\hat{y}$ values, and from these compute the $\hat{k}_y$ values.
- It turns out that this may actually be done without disposing of the Y values.

Outline of the proposed approach

- Extend tree induction algorithm to kernelized output spaces, to exploit the available data (X, K_y) so as to regress an output space feature vector $\hat{y}_t(x)$ in line with the target kernel.
- From $\hat{y}_t(x) = \sum_{i=1} y^i k_t(x^i, x)$ define the target kernel approximation by $\hat{k}_{t,y}(x, x') = \hat{y}_t^T(x)\hat{y}_t(x')$.
- Exploiting the fact that $(y^i)^T y^j = k_y^{i,j}$, express

$$\hat{k}_{t,y}(x, x') = \hat{y}_t^T(x)\hat{y}_t(x') = \sum_{i=1}^{N}\sum_{j=1}^{N} k_t(x^i, x) k_y^{i,j} k_t(x', x^j),$$

to get a model that depends only on the data (X, K_y).

Regression tree induction in kernelized output spaces

- We first adapt the calculation of output space square loss so as to use only the output space kernel values
  - We have:

$$\begin{aligned} \mathrm{var}\{y|s\} &= \frac{1}{\#s}\sum_{i\in s}(y^i - \frac{1}{\#s}\sum_{i\in s} y^i)^2 \\ &= \#s^{-1}\sum_{i\in s} k_y^{i,i} - \#s^{-2}\sum_{i\in s}\sum_{j\in s} k_y^{i,j}. \end{aligned}$$

  - Using this formula, one can grow tree structures to greedily reduce square loss in the output-feature space while only using the given kernel values
  - (May also serve for pruning and getting variable importances)

- NB: If we would like to label leaf nodes in order to make predictions over $\mathcal{Y}$, we would need to solve locally the pre-image problem, but we do not need to do this for making output space kernel predictions.

Kernel regression trees: nature of the approximator

- Over the sample $\hat{k}_{t,y}^{k,l} = \sum_{i=1}^{N} \sum_{j=1}^{N} k_t(x^i, x^k) k_y^{i,j} k_t(x^l, x^j)$, or in matrix form $\hat{K}_{t,y} = K_t K_y K_t$.
- If the sample is sorted according to leaf indices, we have

$$K_t = \begin{bmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & A_\ell \end{bmatrix} \quad \hat{K}_{t,y} = \begin{bmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,\ell} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,\ell} \\ \vdots & \vdots & \ddots & \vdots \\ B_{\ell,1} & B_{\ell,2} & \cdots & B_{\ell,\ell} \end{bmatrix}$$

  where the blocks A_i and $B_{i,j}$ are constant matrices defined respectively for each leaf and for each pair of leaves.
- In other words, $A_i = \alpha_i \mathbf{1} \in \mathbb{R}^{N_i \times N_i}$ and $B_{i,j} = \beta_{i,j} \mathbf{1} \in \mathbb{R}^{N_i \times N_j}$.

Kernel regression trees: nature of the approximator

- The standard tree kernel uses $\alpha_i = N_i^{-1}$, and hence we have

$$\beta_{i,j} = N_i^{-1} N_j^{-1} \sum_{i \in L_i} \sum_{j \in L_j} k_y^{i,j}.$$

  In other words, $\hat{k}_{t,y}(o, o')$ is computed by propagating o and o' in the tree and then averaging the values of k_y over the pairs of subsamples at the reached leaves.

- Frobenius norm optimality of this approximation: we have

$$\hat{K}_{t,y} = \arg\max_{Q \propto t} \sum_{i=1}^{N} \sum_{j=1}^{N} ([K_y]_{i,j} - [Q]_{i,j})^2.$$

  where we denote by $\{Q \propto t\}$ the set of Gram matrices depending only on the partition induced by t.

Kernel regression trees: effect of splitting nodes

- Splitting a leaf in the tree, replaces the corresponding line and column of blocks in $\hat{K}_{t,y}$ by two lines and two columns of smaller blocks.
- Splitting a leaf hence decreases Frobenius norm monotonically.
- Fully grown trees yield $N \times N$ blocks of size 1, and hence a Frobenius norm of zero.

* Comment about optimal splitting rules:
  - The classical optimal splitting rule seeks to reduce variance in the output feature space, which may be evaluated from the sole information of the diagonal block corresponding to the node being split.
  - The above reasoning suggests to adapt the optimal splitting rule when approximating kernels in terms of Frobenius norm (at the price of higher computational complexity).

Extending to ensembles of trees

We are currently studying two averaging schemes, to extend to ensembles of randomized trees

- Averaging of the individual kernel approximations:

$$\overline{k}_{\mathcal{T},y}(o,o') = M^{-1}\sum_{k=1}^{M}\hat{k}_{t_k,y}(o,o').$$

- Convoluting $k_{\mathcal{T}}(o,o') = M^{-1}\sum_{k=1}^{M} k_{t_k}(o,o')$ with the k_y:

$$\underline{k}_{\mathcal{T},y}(o,o') = \sum_{i=1}^{N}\sum_{j=1}^{N} k_{\mathcal{T}}(o_i,o)k_y(o_i,o_j)k_{\mathcal{T}}(o',o_j).$$

NB: these ideas may be extended to gradient boosting of trees.

Summary

- Tree induction methods define a local averaging scheme to infer an approximation of a kernel given a Gram matrix of this kernel together with an input space feature representation.
- Splitting nodes may use an efficient proxy based on variance reduction in an induced output space in the context of Frobenius norm kernel loss. Other loss functions may be used.
- The main specificity of the tree based approach is that the kernel over the input space is directly inferred from the available data set (X, K_y). In this context, feature selection and importance evaluation abilities may be of interest.
- Computational complexity is in the order of $nN^2 \log N$ where n is the dimension of the input space and N the sample size.

Exploiting the kernel view of tree-based methods

- Input space kernel induced by an ensemble of trees
- Supervised learning of (output space) kernels
- Semi-supervised learning and censored data

Semi-supervised learning and censored data

- Censored data: for some i/o pairs the value of $y(o)$ is upper/lower bounded, but otherwise unknown
- Semi-supervised learning: for some i/o pairs the value of $y(o)$ is missing
- Handle these two problems by regularizing tree-based predictions by using convex optimization

Generic formulation

- Denote by $Y \in \mathbb{R}^N$ the vector of sample outputs, by $Z \in \mathbb{R}^P$ the vector of all leaf labels of an ensemble $\mathcal{T}$, by $K_{\mathcal{T}} \in \mathbb{R}^{N \times N}$ its gram matrix, by $L_{\mathcal{T}} \in \{0, M^{-1}\}^{N \times P}$ its partitioning matrix, by $\nu \in \mathbb{R}^N$ a vector of slack variables, and by $\Omega(\cdot,\cdot,\cdot) : \mathbb{R}^N \times \mathbb{R}^P \times \mathbb{R}^N \to \mathbb{R}$:
- Consider the following optimization problem:

$$\min \Omega(\Delta_Y, \Delta_Z, \nu) \;\; s.t.$$
$$|\, K_{\mathcal{T}} Y + K_{\mathcal{T}} \Delta_Y + L_{\mathcal{T}} \Delta_Z - Y \,| \leq \nu$$
$$(\Delta_Y, \Delta_Z, \nu) \in C \subset \mathbb{R}^{N+P+N}$$

- This problem allows to compute *adjustments* Δ_Y to the sample output values and/or Δ_Z to the labels associated to leaves, while minimizing the *regularization* term Ω and respecting some additional constraints C. We impose that Ω and C are convex.

Supervised learning with censored data

Problem formulation

- In addition to a sample $ls = (X, Y)$, with precise output values, we dispose of a sample $ls_c = (X_c, Y_c)$ where output information is censored, e.g. specified in terms of intervals in the form $[\underline{y}^i, \overline{y}^i]$.
- We want to exploit both ls and ls_c to infer a tree based predictor.

Strategy

- First use the standard sample ls to infer an ensemble of tree structures and corresponding kernel K.
- Second, use convex regularization framework, to refit labels Y and Z while taking into account the information in ls_c.

Example: formulation for survival data

- In survival data, the incomplete information corresponds to cases which left the study before "dying", hence their output (survival time) is only given in the form of a vector of lower bounds $\underline{Y}_c$.
- We propose the following regularization scheme:

$$\begin{array}{ll} \min & C_1\|\Delta_Y\| + C_2\|\Delta_Z\| + C_3\|\nu\| + C_4\|\nu_c\| \\ \text{s.t.} & -\nu \leq KY + K\Delta_Y + L\Delta_Z - Y \leq \nu \\ & -\nu_c \leq K_cY + K_c\Delta_Y + L_c\Delta_Z - \underline{Y}_c \end{array}$$

- This formulation aims at refitting the model, so as to satisfy constraints imposed by the censored data (in a soft way controlled by the regularization term $C_4\|\nu_c\|$).

Semi-supervised learning

Problem formulation

- In addition to a sample $ls_l = (X_l, Y_l)$, with precise output labels, we dispose of an unlabeled sample $ls_u = (X_u)$ where output information is unknown.
- We want to exploit both ls_l and ls_u to infer a tree based predictor.

Strategy

- First use the standard sample ls_l to infer an ensemble of tree structures and compute the corresponding gram matrix K' (using the complete set of inputs $X' = (X_l, X_u)$).
- Second, use convex regularization framework, to find Y_u while taking into account the information in ls_l and ls_u.

Example: formulation for manifold regularization

- In manifold regularization, we exploit a graph $\mathcal{L}$ expressing proximities among objects, and we want objects that are close according to the graph to yield similar predictions.
- To this end, we may immediately transpose the following regularization scheme towards ensembles of trees:

$$\begin{array}{ll} \min & \|\nu\|_2^2 + C(Y' + \Delta_{Y'})^T K' \mathcal{L} K' (Y' + \Delta_{Y'}) \\ \text{s.t.} & -\nu \leq \left(K'_{l,l} | K'_{l,u}\right)(Y' + \Delta_{Y'}) - Y_l \leq \nu, \\ & \Delta_{Y_l} = 0; \, Y_u = 0. \end{array}$$

- This classical(M. Belkin et al, JMLR, 2006) formulation aims at computing labels for the unlabelled objects, so that nearby objects yield similar output predictions and so that the known outputs are not too much different from their sample values.

Summary

- We have proposed a combination of primal/dual formulations to regularize tree-based models by exploiting convex (in practice linear or quadratic) optimization solvers
- The proposed formulation puts supervised, semi-supervised and censored problems in a same framework, and allows to exploit various ideas proposed in these contexts to tree-based methods.
- Further possibilities
  - Combination of semi-supervised and censored data
  - Exploiting prior knowledge (symmetries, constraints among multiple outputs, monotonicity relations etc.)
- Our first experiments show promising results on various applications.

Further reading about kernel view of regression trees

P. Geurts, L. Wehenkel, and F. d'Alché-Buc.
Kernelizing the output of tree-based methods.
In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 345-352 - 2006.

P. Geurts, L. Wehenkel, and F. d'Alché-Buc.
Gradient boosting for kernelized output spaces.
In *Proceedings of the 24th International Conference on Machine Learning*, pp. 289-296 - 2007.

B. Cornélusse et al.
Tree-based ensemble model regularization by convex optimization.
Submitted, 2009.

P. Geurts et al.
Kernel regression with tree-based methods.
Technical report, to be submitted, 2009.

Part III

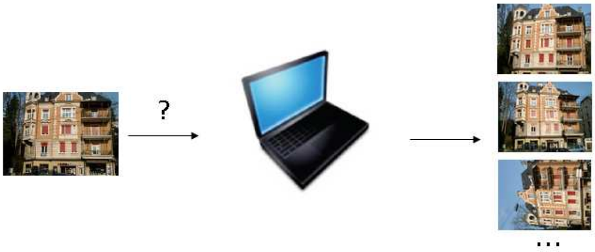
Handling structured input spaces

Content-based image retrieval

- Goal
  - Given a database of *unlabeled* reference images $\{I_q\}$, build a model able to retrieve the reference images most similar to a new query image I_r based only on visual content.

Content-Based Image Retrieval

- Goal
  - Given a database of *unlabeled* reference images $\{I_q\}$, build a model able to retrieve the reference images most similar to a new query image I_r based only on visual content.



Outline of the Segment & Combine approach

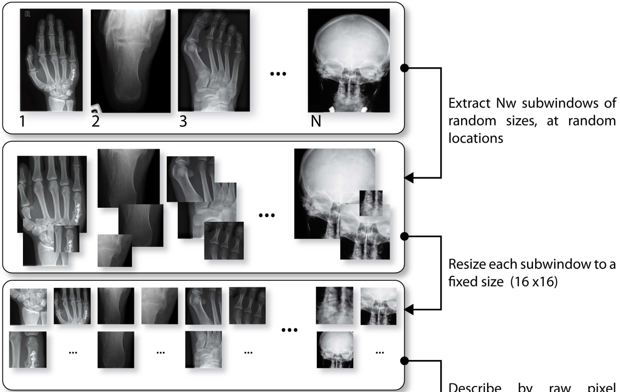
Define kernel among images I_q and I_r by

$$k_I(I_q, I_r) = \frac{1}{\#S(I_q)\#S(I_r)} \sum_{(s,s')\in S(I_q)\times S(I_r)} k_S(s, s')$$

where $S(\cdot)$ is a subwindow extractor and $k_S(\cdot,\cdot)$ a subwindow kernel.

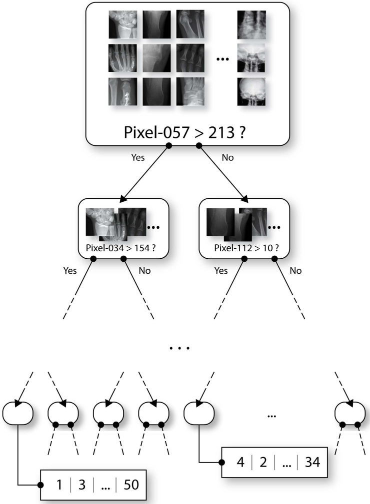
- To extract subwindows from an image
  - pick a fixed number of them of random location and shape.
- To learn the kernel $k_S(\cdot,\cdot)$
  - use a dataset of subwindows extracted from reference images
  - represent them by a vector of constant dimension (e.g. pixels)
  - build a tree ensemble on this representation
- Use the trees ensemble
  - as a subwindow indexing scheme, in order to efficiently compute $k_I(I_q, I_r)$ for a given query image and $\forall I_r$, and collect the top ranked ones.

Step 1: Extraction of random reference image subwindows



	Pixel-001	Pixel-002		Pixel-256
Subwindow-00	34	201		255
Subwindow-01	78	31		204
Subwindow-02	145	167		32
...				
Subwindow-Nw	14	4		134

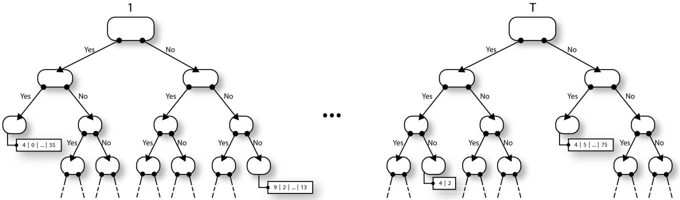
Step2: Indexing structure (one tree)



With a single tree, the kernel among a query image and a reference image is only determined by the counts of the number of their subwindows reaching common leafs (modulo the total number of reference image subwindows reaching that leaf).

Thus, to remember the relevant information about the subwindows of the reference images needed to compute the image kernel, we need only to maintain at each leaf a sparse list of the number of the subwindows of each reference image reaching that leaf.

Step2: Indexing structure (an ensemble of trees)



- Parameters
  - M: the number of totally randomized trees
  - n_{min}: the minimum node size, stop-splitting of a node if $size(node) < n_{min}$
- Complexity
  - $O(MN\log_2(N))$ in the total number of subwindows (independent of the dimensionality of the feature space)

Step 3: Image retrieval given query image

- Decompose the query image I_q into its subwindows by $S(\cdot)$.
- Propagate each subwindow in each tree
  - at a leaf, exploit the counts of subwindows for each reference image by cumulating this information (modulo the number of reference subwindows reaching each leaf) to compute the ensemble kernel between I_q and each reference image I_r.
- In the end, all reference images are ranked by decreasing order of this kernel and the top ranking ones may be retrieved.
- NB: the scheme may also be distributed over a set of image databases which are stored at remote locations, by imposing that they use a common ensemble of randomized trees to index the subwindows of their local image repositories.

IRMA (1/3): X-Ray images (from http://irma-project.org/)

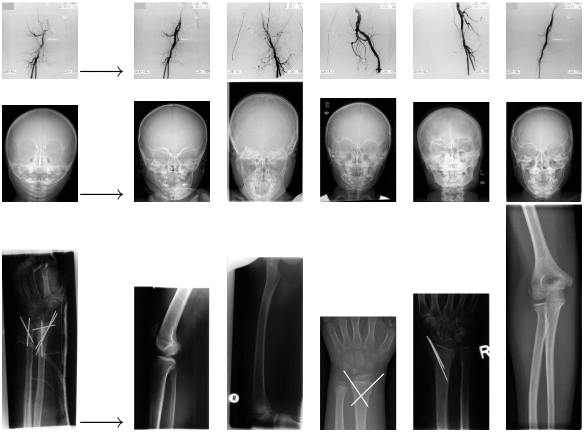
IRMA (2/3): Results

- Protocol
  - 9000 *unlabeled* reference images (approx. 512×512)
  - 1000 *labeled* test images (57 classes)
  - Recognition rate of the first ranked image
- Results

Dataset	ls/ts	us	naïve	NN	KDGN07
IRMA	9000/1000	85.4%	29.7%	63.2%	87.4%

(with $M = 10$ trees, $nw = 1000$ subwindows per image, $n_{\text{min}} = 2$, ie. fully developed trees)

IRMA (3/3): query $\longrightarrow$ top 5 retrieved images



Summary

- The sub-window extraction scheme controls the nature of the invariances injected into the learning algorithm.
- For content based image retrieval, we use totally randomized tree ensembles.
- The approach is very generic, and is able at the same time to yield state-of-the-art results on a diversity of image retrieval problems.
- The approach may be casted in a distributed context, where image similarities among distant servers are aligned and may be computed locally, yielding a highly scalable and distributed CBR approach.

Other image analysis applications

- Image classification
- Image segmentation
- Image annotation

Other kinds of structured inputs

- Time-series as inputs
- Sequences as inputs
- Graphs as inputs

Essence of the S&C framework

- The segmentation and combination schemes may be tailored to the kind of invariances and degree of locality that are supposed to be present in a learning problem.
- In general, the larger the number of segments that are used (both at learning and at prediction times), the better the results.
- The involved algorithms are amenable to parallel computations, and with some modifications they may also be casted in a data distributed fashion.

Some references for further reading

R. Marée, P. Geurts, J. Piater, and L. Wehenkel.
Random subwindows for robust image classification.
In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, Vol. 1, pp. 34-40 - June 2005.

R. Marée, P. Geurts, and L. Wehenkel.
Content-based Image Retrieval by Indexing Random Subwindows with Randomized Trees
In *ACCV 2007 / IPSJ Transactions on Computer Vision and Applications*, Vol. 1, No 1, pp. 46-57 - Jan 2009

R. Marée et al.
Incremental Indexing and distributed image search using shared randomized vocabularies.
Submitted, 2009.

Part IV

Wrap up

The multiple roles of randomization

- Randomization is useful to regularize learning algorithms in various ways
- Randomization is also useful to reduce the computational complexity of optimization algorithms
- Randomization is also a way to enforce invariances in learning and optimization problems

- *It seems that looking at all these aspects simultaneously may open new directions for the design of learning and optimization algorithms.*

Other related topics

- Extensions to boosting
- Generic ways of handling missing data
- Generic ways of handling invariances
- Generic ways of handling complex input and output spaces

Ongoing works

We work on applications in (mainly)

- Image analysis
  - in particular, exploiting biomedical images
- Bioinformatics
  - in particular, biomarker identification, biological network inference, exploiting macro-molecule structures
- Electric energy systems
  - in particular, large scale power systems' security management and optimal scheduling of energy resources

We work on methods also in

- Multi-stage decision problems under uncertainty
  - reinforcement learning, stochastic programming
- Unsupervised learning
  - with ensembles of tree structured graphical probabilistic models

Acknowledgments

This work was done in collaboration with:

- Bertrand Cornélusse, Boris Defourny, Damien Ernst, Pierre Geurts, Raphaël Marée, Justus Piater, University of Liège
- Florence d'Alché-Buc, University of Evry val d'Essonne

with the support of:

- FNRS Belgium and CNRS France
- The Networks DYSCO and BIOMAGNET, funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office.
- PASCAL2, Network of Excellence funded by the European Commission.