

Gradient Boosting for Kernelized Output Spaces

Pierre Geurts^{1,2}
Louis Wehenkel²
Florence d'Alché-Buc¹

P.GEURTS@ULG.AC.BE
L.WEHENKEL@ULG.AC.BE
DALCHE@IBISC.UNIV-EVRY.FR

¹IBISC FRE CNRS 2873, University of Evry, 523, Places des Terrasses, 91 Evry, France, ²Department of EECS & GIGA-R, University of Liège, Sart Tilman, B28, B-4000, Liège, Belgium

Abstract

A general framework is proposed for gradient boosting in supervised learning problems where the loss function is defined using a kernel over the output space. It extends boosting in a principled way to complex output spaces (images, text, graphs etc.) and can be applied to a general class of base learners working in kernelized output spaces. Empirical results are provided on three problems: a regression problem, an image completion task and a graph prediction problem. In these experiments, the framework is combined with tree-based base learners, which have interesting algorithmic properties. The results show that gradient boosting significantly improves these base learners and provides competitive results with other tree-based ensemble methods based on randomization.

1. Introduction

Prediction in structured output spaces has recently emerged as a key challenging problem in statistical learning. The availability of structured data sets from XML documents to biological structures, such as sequences or graphs, has prompted for the development of new learning algorithms able to handle complex structured output spaces. Among recent solutions proposed in the literature, one can distinguish classification and regression based solutions (Cortes et al., 2005). Loosely speaking, classification based approaches exploit a feature space defined on input-output pairs and formulate the problem as that of finding a linear function in this feature space that provides a high score to those input-output pairs appear-

ing in the learning sample while maximizing a margin (Tsochantaridis et al., 2005; Taskar et al., 2005). Regression based approaches, on the other hand, embed the output space into a kernel induced Hilbert space and compute a mapping from inputs towards this latter space by exploiting the kernel trick in standard regression methods (Weston et al., 2002; Cortes et al., 2005; Szedmak et al., 2005). Predictions in the original output space are then obtained by solving the so-called pre-image problem. Notice that, while the exploitation of a joint feature space is potentially an advantage of classification based approaches for certain applications, they usually have a higher computational cost than regression ones (see Memisevic, 2006 for an introduction to both approaches and Cortes et al., 2005 for a comparison of them).

A recent example of regression based approaches can be found in (Geurts et al., 2006b) where regression trees are extended to kernelized output spaces. The experiments described in that paper suggest that in terms of accuracy this approach is specially effective in the context of ensembles of randomized trees. In the present paper, we investigate how boosting can be generalized to deal with structured output spaces. Boosting, originally introduced by Freund and Schapire (1997) in the context of classification, has been shown to offer a general framework to enhance any weak learner, in particular when considering its interpretation as a gradient descent in a functional space (Friedman, 2001; Mason et al., 2000). Its success in many application areas makes it a natural candidate also for learning in kernelized output spaces.

The rest of the paper is organized as follows. In Section 2, we present a new boosting algorithm, called OKBoost, which extends least squares gradient boosting (Friedman, 2001) to kernelized output spaces. We first derive a general form of the algorithm and then specialize it for tree-based base learners which are specially attractive from the algorithmic point of view. In Section 3, this combination is evaluated on three

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007. Copyright 2007 by the author(s)/owner(s).



problems where it shows significant improvement with respect to a single trees and provides competitive results with other tree based ensemble methods. Section 4 concludes and discusses future work.

2. Gradient Boosting for Prediction in Kernelized Output Spaces

2.1. Least Squares Gradient Boosting

The general problem of supervised learning can be formulated as follows: from a learning sample $\{(x_i, y_i)\}_{i=1}^N$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, find a function $F : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes the expectation of some loss function ℓ over the joint distribution of input/output pairs, $E_{x,y}\{\ell(F(x), y)\}$.

The idea of boosting is to approximate F by building sequentially a linear combination of “weak” learners in such a way that, at each step, more importance is given to data that have been unsuccessfully learned at the previous steps. We focus here on the regression setting and consider the gradient boosting framework proposed by Friedman (Friedman, 2001). Let us first suppose that $\mathcal{Y} = \mathbb{R}$ and that we want to find an approximation $F(x)$ in the form:

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

where F_0 is some initial approximation, $\{h(x; a)\}_{a \in A}$ is a set of basis functions mapping inputs to \mathbb{R} , and $\beta_m \in \mathbb{R}$. In this context, boosting builds the model $F(x)$ in a greedy-stagewise way, by adding at each step a new basis function to the approximation chosen so as to reduce as much as possible the empirical loss. For $m = 1, 2, \dots, M$, this translates into:

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

$$F_m(x) = F_{m-1}(x) + \beta_m h(x; a_m). \quad (2)$$

Since (1) may be difficult to compute, the idea developed in (Friedman, 2001) is to compute instead the function $h(x; a_m)$ that is the most parallel to the steepest-descent direction in the N -dimensional data space at $F_{m-1}(x)$. With a square-error loss function, this procedure reduces to the (least-squares) residual fitting procedure given in Algorithm 1 (see Friedman, 2001), where we assume that the scaling parameter β is incorporated in the base-learner¹ and that F_0 is the sample average of the outputs.

¹This is the case of regression trees (and subsequently of output kernel trees) since leaf predictions are chosen to minimize square error.

Algorithm 1 Gradient boosting with square loss

```

 $F_0(x) = \frac{1}{N} \sum_{i=1}^N y_i$ 
for  $m = 1$  to  $M$  do
  (a)  $y_i^m = y_i - F_{m-1}(x_i), i = 1, \dots, N$ 
  (b)  $a_m = \arg \min_a \sum_{i=1}^N (y_i^m - h(x_i; a))^2$ 
  (c)  $F_m(x) = F_{m-1}(x) + h(x; a_m)$ 
end for

```

Algorithm 2 Abstract gradient boosting with square loss in output feature space

```

 $F_0^\phi(x) = \frac{1}{N} \sum_{i=1}^N \phi(y_i)$ 
for  $m = 1$  to  $M$  do
  (a)  $\phi_i^m = \phi(y_i) - F_{m-1}^\phi(x_i), i = 1, \dots, N$ 
  (b)  $a_m = \arg \min_a \sum_{i=1}^N \|\phi_i^m - h^\phi(x_i; a)\|^2$ 
  (c)  $F_m^\phi(x) = F_{m-1}^\phi(x) + h^\phi(x; a_m)$ 
end for

```

Algorithm 2 can be easily extended to multiple outputs (i.e., $\mathcal{Y} = \mathbb{R}^n$), by using the euclidean loss function $\ell(y_1, y_2) = \|y_1 - y_2\|^2$ and replacing step (b) by:

$$a_m = \arg \min_a \sum_{i=1}^N \|y_i^m - h(x_i; a)\|^2.$$

It can thus be applied to any base learner handling multiple outputs with a euclidean loss function.

2.2. Output-Kernel Based Gradient Boosting

Let us suppose now that we have a kernelized output space, i.e. an output space \mathcal{Y} endowed with a kernel $k : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$, and let us denote by $\phi : \mathcal{Y} \rightarrow \mathcal{H}$ the feature map defined by k . Substituting outputs y in Algorithm 1 by vectors $\phi(y) \in \mathcal{H}$, we get Algorithm 2 where h_ϕ and F_ϕ now denote functions from \mathcal{X} to \mathcal{H} and residuals ϕ_i^m belongs to \mathcal{H} . Notice that this algorithm minimizes the loss:

$$\begin{aligned} \ell(y_1, y_2) &= \|\phi(y_1) - \phi(y_2)\|^2 \\ &= k(y_1, y_1) + k(y_2, y_2) - 2k(y_1, y_2). \end{aligned} \quad (3)$$

which depends only on the output kernel. Thus, to exploit this idea in practice, assuming that the kernel values are given over the learning sample, we need to be able to learn functions $F_0^\phi(x)$ and $h^\phi(x; a_m)$ from kernel values and to define a way to compute pre-images $\hat{y} = \phi^{-1}(F_M^\phi(x))$ from the final model. We address these two problems separately below.

2.2.1. LEARNING STAGE (FROM KERNEL VALUES)

We assume that the base learner can handle a kernelized output space, i.e. that it takes only as input kernel values between updated outputs $k_{i,j}^m = \langle \phi_i^m, \phi_j^m \rangle$. Hence, during the training stage, we need to compute

Algorithm 3 Output kernel based boosting (learning stage)

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: $w_i(x; a_0) \equiv 1/N$, $W_{i,j}^0 = 1/N, \forall i, j = 1, \dots, N$, $K^0 = K$.
 - 2: **for** $m = 1$ **to** M **do**
 - 3: $K^m = (I - W^{m-1})^T K^{m-1} (I - W^{m-1})$
 - 4: Apply the base learner to the output Gram matrix K^m to get a model $(w_i(x; a_m))_{i=1}^N$.
 - 5: Compute $W_{i,j}^m = w_i(x_j; a_m), i, j = 1, \dots, N$ from the resulting model.
 - 6: **end for**
-

these Gram matrices for $m = 1, \dots, M$ from the knowledge of kernel values between learning sample outputs and the model obtained at the previous step.

One situation where this is achievable is when predictions in the output feature space given by the base learner can be written as linear combinations of output feature space vectors corresponding to instances from the learning sample. Indeed, let us assume that the predictions given by any basis function $h^\phi(x; a_m)$ from the sequence may be written as:

$$h^\phi(x; a_m) = \sum_{i=1}^N w_i(x; a_m) \phi_i^m, \quad (4)$$

where the weights w_i of the combination are (parametrized) functions from the input space \mathcal{X} to \mathbb{R} . In this case, the kernel values $k_{i,j}^m$ can be computed by (by application of Eqn (2)):

$$\begin{aligned} k_{i,j}^m &\triangleq \langle \phi_i^m, \phi_j^m \rangle \\ &= \langle \phi_i^{m-1} - h^\phi(x_i; a_{m-1}), \phi_j^{m-1} - h^\phi(x_j; a_{m-1}) \rangle \end{aligned}$$

Taking Eqn (4) into account and replacing $\langle \phi_i^{m-1}, \phi_j^{m-1} \rangle$ by $k_{i,j}^{m-1}$ this yields:

$$\begin{aligned} k_{i,j}^m &= k_{i,j}^{m-1} - \sum_{l=1}^N w_l(x_j; a_m) k_{i,l}^{m-1} - \sum_{l=1}^N w_l(x_i; a_m) k_{l,j}^{m-1} \\ &\quad + \sum_{k,l=1}^N w_k(x_i; a_m) w_l(x_j; a_m) k_{k,l}^{m-1}, \end{aligned}$$

which uses only kernel values between outputs from the previous iteration. Putting all these ideas together yields (in matrix notation²) Algorithm 3, which starts from the original output Gram matrix $K^0 = K$ and iterates the application of the base learner to updated Gram matrices to yield the weight functions $w_i(x, a_m)$ and the Gram matrix for the next step. Note that, given the initial approximation F_0^ϕ , the computation of K^1 amounts at centering the data in the output feature space.

²where the superscript T denotes matrix transposition.

2.2.2. PREDICTION STAGE

Algorithm 3 computes a sequence of weight functions $w_i(x; a_m)$ represented by their parameters a_m . The final model is computed by (2) combined with (4). As it is a linear combination of basis functions, themselves linear combinations of output feature vectors, it may be written as:

$$F_M^\phi(x) = \sum_{i=1}^N w_i^F(x) \phi(y_i). \quad (5)$$

Indeed, denoting by $p^m(x)$ the (line) vector $(w_1(x; a_m), \dots, w_N(x; a_m))$ of weights obtained by training the m -th model (cf Eqn 4) and posing $p^0(x) = (1/N, \dots, 1/N)$, it is straightforward to show that in Eqn (5) the vector $w^F(x) = (w_1^F(x), \dots, w_N^F(x))$ for some input x is obtained by

$$w^F(x) = \sum_{m=0}^M p^m(x) O^m, \quad (6)$$

where the $N \times N$ matrices O^m are computed recursively by:

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

in which I denotes the identity matrix and W^{m-1} the matrices already defined in Algorithm 3.

From the model (5), we can make two kinds of inferences: predictions in the original output space \mathcal{Y} and kernel values over $\mathcal{Y} \times \mathcal{Y}$. The prediction of kernel values given inputs may be obtained by using only the original output Gram matrix K over the learning sample:

$$\hat{k}(x, x') = \langle F_M^\phi(x), F_M^\phi(x') \rangle = w^F(x) K (w^F(x'))^T. \quad (7)$$

The prediction of outputs in \mathcal{Y} , given the kernel k , may be derived by solving the following pre-image problem (from Eqn 5):

$$\begin{aligned} F_M^\mathcal{Y}(x) &= \arg \min_{y' \in \mathcal{Y}} \|\phi(y') - F_M^\phi(x)\|^2 \\ &= \arg \min_{y' \in \mathcal{Y}} \left\{ k(y', y') - 2 \sum_{i=1}^N w_i^F(x) k(y_i, y') \right\}. \end{aligned} \quad (8)$$

Algorithm 4 Computing output and kernel predictions

input a test sample of Q input vectors, $\{x'_1, \dots, x'_Q\}$.

output a prediction $F_M^{\mathcal{Y}}(x'_i) \in \mathcal{Y}$ for each input $x'_i, i = 1, \dots, Q$ and an output kernel matrix prediction \hat{K} with

$$\hat{K}_{i,j} = \langle F_M^\phi(x'_i), F_M^\phi(x'_j) \rangle, i, j = 1, \dots, Q$$

- 1: $O^0 = I, W_{i,j}^0 = 1/N, \forall i, j = 1, \dots, N, W_{i,j}^F = \frac{1}{N}, \forall i = 1, \dots, Q, j = 1, \dots, N$
 - 2: **for** $m = 1$ **to** M **do**
 - 3: $O^m = O^{m-1} - (W^{m-1})^T O^{m-1}$
 - 4: Compute the $Q \times N$ matrix P^m with $P_{i,j}^m = w_j(x'_i; a_m), \forall i = 1, \dots, Q, \forall j = 1, \dots, N$
 - 5: Set W^F to $W^F + P^m O^m$
 - 6: Compute $W_{i,j}^m = w_i(x_j; a_m), \forall i, j = 1, \dots, N$ from the m th model
 - 7: **end for**
 - 8: Compute $S = 1_{Q \times 1} \text{diag}(K)' - 2W^F K$.
 - 9: $F_M^{\mathcal{Y}}(x'_i) = y_k$ with $k = \arg \min_{j=1, \dots, N} S_{i,j}, \forall i = 1, \dots, Q$ // **Pre-image computation**
 - 10: $\hat{K} = W^F K (W^F)^T$ // **Kernel predictions**
-

If k is unknown (except over the learning sample), or for computational reasons, the arg min over \mathcal{Y} in Eqn (8) may be replaced by an arg min over the outputs of the learning sample (Weston et al., 2002).

In Eqn (8) (resp. (7)) the main point is the computation of the weight vector $w^F(x)$ (resp. $w^F(x)$ and $w^F(x')$). Since matrices $O^m, m = 1, \dots, M$ are of dimension N^2 , it is usually not practical to precompute and store them. They then need to be recomputed each time predictions are needed, but if we know in advance a set of inputs for which we want to make predictions, we can compute them only once and make predictions simultaneously for all instances. This is the purpose of Algorithm 4. Note that in line 8-9 of this algorithm the arg min of Eqn (8) is approximated by computing it only over outputs of the learning sample. Although generic and simple, this approximation is also restrictive and could certainly affect generalization. However, our algorithm can in principle exploit other techniques for computing pre-images, in particular efficient exact solutions tailored to a particular output-space/kernel combination (e.g., the algorithm for string kernels of Cortes et al., 2005). Since Algorithm 3 does not require pre-image computations, other pre-image solvers can be exploited by merely plugging them in replacement of line 8-9 in Algorithm 4. Finally, note also that no pre-image computations is required to make kernel predictions (line 10).

2.3. Regularization

To avoid overfitting, the number of terms M of the model could be adapted to the problem at hand, e.g. by cross-validation. Alternatively one can use shrinkage, as advocated in Friedman (2001). Shrinkage replaces the update rule (c) in Algorithm 1 by:

$$F_m(x) = F_{m-1}(x) + \nu h(x; a_m), 0 < \nu \leq 1 \quad (9)$$

where ν is a learning rate kind of parameter. The introduction of this parameter in the kernelized algorithms of Algorithms 3 and 4 is straightforward.

Another approach to avoid overfitting is to introduce some randomization into the procedure, e.g. by learning the base learner from a random subsample of the training data (Friedman, 2002). In our experiments with tree-based methods, we will exploit a specific randomization technique (see Section 2.4.2).

2.4. Tree-Based Base Learner

While boosting algorithms can be used with any base learner, they have been mainly used in conjunction with decision or regression trees and in particular with stumps (i.e. trees reducing to one split). In the context of our kernelized boosting algorithm, the base learner should be able to optimize square error in a kernelized output space and provide predictions in the form (4). In a recent paper, Geurts et al. (2006b) proposed a kernelization of the output of regression tree based algorithms, called OK3 (for output kernel trees), that satisfies these two conditions. We briefly describe this method before going through specific details related to its use in the context of gradient boosting.

2.4.1. OUTPUT KERNEL TREES (OK3)

Standard regression trees recursively partition the learning sample by selecting a split based on an input variable reducing as much as possible the variance of the output in the left and right subsamples of learning cases. OK3 proceeds in the same way while using a score computed in the feature space \mathcal{H} induced from the kernel k by:

$$\text{Score}(T, S) = \text{var}\{\phi|S\} - \frac{N_l}{N} \text{var}\{\phi|S_l\} - \frac{N_r}{N} \text{var}\{\phi|S_r\},$$

where T is the split to evaluate, S is the local learning sample of size N at the node to split, S_l and S_r are its left and right successors of size N_l and N_r respectively, and $\text{var}\{\phi|S\}$ denotes the variance of output feature vectors in the subset S defined by:

$$\text{var}\{\phi|S\} = \frac{1}{N} \sum_{i=1}^N \|\phi_i - \frac{1}{N} \sum_{j=1}^N \phi_j\|^2. \quad (10)$$

OK3 exploits the fact that this variance may be expressed only in terms of output-kernel values:

$$\text{var}\{\phi|S\} = \frac{1}{N} \sum_{i=1}^N k_{i,i} - \frac{1}{N^2} \sum_{i,j=1}^N k_{i,j}. \quad (11)$$

Once a tree is grown, the prediction made by OK3 in the output feature space \mathcal{H} corresponding to a leaf L is in principle the center of mass in the leaf:

$$\hat{\phi}_L = \frac{1}{N_L} \sum_{i=1}^{N_L} \phi_i, \quad (12)$$

where $\{\phi_1, \dots, \phi_{N_L}\}$ are the outputs (in \mathcal{H}) reaching the leaf. To obtain a prediction in \mathcal{Y} from (12), Geurts et al. (2006b) propose to restrict the search of the pre-image to the outputs that appear in the leaf.

2.4.2. GRADIENT BOOSTING WITH OK3

The use of output kernel trees in Algorithm 3 is straightforward. The OK3 algorithm is fed with the current output kernel matrix K^m and returns a tree which is then used to compute the weight matrix W^m . The prediction of a tree at some point x is given by expression (4) with $w_i(x; a_m) = 1/N_L$ if x and x_i reach the same leaf of size N_L , 0 otherwise. With this definition of the weights, the matrices W^m are thus symmetric and positive definite. Indeed, each element $W_{i,j}^m$ corresponds to a dot-product in a feature space defined on \mathcal{X} such that $\phi(x)$ 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. During the prediction stage, matrices P^m are simply obtained by propagating the Q test inputs through the m trees taking $P_{i,j}^m$ equal to $1/N_L$ if x'_i reaches the leaf L that contains x_j , 0 otherwise.

For this OK3 based boosting algorithm to work well, we need to constrain the complexity of the trees. Following Friedman (2001), we propose to use trees with at most J splits (or $J+1$ terminal nodes). In order to grow such trees, we use an approximate best first strategy, which consists, at each step of the induction, in splitting the node of highest total variance $N \times \text{var}\{\phi(y)|S\}$. J is thus a meta-parameter of the

algorithm which optimal value should reflect the interaction order of the input variables in the (unknown) target function (Friedman, 2001).

In addition to standard OK3, we will also use as base learner randomized output kernel trees. The motivation for this is that randomizing the base learner should slow down convergence and reduce the risk of overfitting by decreasing learning variance. We hope that in combination with the bias reduction of boosting, this will lead to better results than using standard OK3. We use the randomization of Geurts et al. (2006a): an extremely randomized tree (Extra-tree) is grown from the full learning sample by selecting at each tree node the split of maximum score among K splits obtained by drawing an input variable and a cut-point (or a subset of values, in the symbolic case) at random. In our experiments, we will use the default value of the parameter K of this algorithm, corresponding to the square root of the dimension of the input space. We will denote by OK3-et an output kernel tree obtained with this randomization.

2.4.3. CONVERGENCE WHEN $m \rightarrow \infty$

Let us show that with tree-based base learners, the empirical error of gradient boosting is non increasing and hence guaranteed to converge (since it is bounded from below by 0). Indeed, assuming no shrinkage ($\nu = 1$), the error at the m th step is given by:

$$\begin{aligned} \sum_{i=1}^N \|\phi(y_i) - F_m(x_i)\|^2 &= \sum_{i=1}^N \|\phi_i^m - h^\phi(x_i; a_m)\|^2 \\ &= \sum_{l=1}^{J+1} \sum_{j=1}^{N_l} \|\phi_{i_j^l}^m - \frac{1}{N_l} \sum_{k=1}^{N_l} \phi_{i_k^l}^m\|^2 \end{aligned} \quad (13)$$

where the outer sum is over tree leaves and the inner sum over the N_l examples that reach leaf l indexed by i_j^l . Since the center of mass is the constant that minimizes the mean squared error in a sample, we have for each leaf l :

$$\sum_{i=1}^{N_l} \|\phi_{i_i^l}^m - \frac{1}{N_l} \sum_{i=1}^{N_l} \phi_{i_i^l}^m\|^2 \leq \sum_{i=1}^{N_l} \|\phi_{i_i^l}^m\|^2,$$

which, from (13), leads to:

$$\begin{aligned} \sum_{i=1}^N \|\phi(y_i) - F_m(x_i)\|^2 &\leq \sum_{i=1}^N \|\phi_i^m\|^2 \\ &= \sum_{i=1}^N \|\phi(y_i) - F_{m-1}(x_i)\|^2. \end{aligned}$$

Convergence actually also holds for $\nu \in [0; 2]$.

2.5. Implementation Issues

In general, the complexities of the update rules in Algorithms 3 and 4 are cubic in the size of the learning sample, as they involve a product of $N \times N$ matrices. In the case where the base learner is OK3 however, the particular nature of matrices W^m makes the complexity of these products only quadratic³ in N . Besides, the development of one node in OK3 is also quadratic in the size of the learning sample, and hence, at fixed tree complexity, this makes gradient boosting with OK3 overall quadratic in the size of the learning sample (and linear in the number of input variables), both for the learning and the prediction stages. This complexity is equivalent to the complexity of single and randomized ensembles of output kernel trees (Geurts et al., 2006b); in practice, we observed higher computing times with boosting, both for learning and for testing. This comes from the fact that boosting can produce very unbalanced trees that require more operations than balanced ones. Boosting also produces much less sparse solutions than single trees or ensembles of randomized trees. In the context of gradient boosting, computing times can thus not be significantly reduced at the prediction stage by reducing the search for the pre-image to the outputs in the support of $w^F(x)$ as suggested in (Geurts et al., 2006b).

3. Experiments

In this section, we apply OKBoost to three problems: **Friedman1** (Friedman, 1991): As a sanity check, we first consider a standard regression problem. This problem is an artificial problem with 10 inputs uniformly drawn between $[0;1]$. The regression function is a non linear function of 5 of these inputs (plus Gaussian noise), while the 5 remaining variables are irrelevant. We use a linear output kernel. In this case, output kernel trees are thus strictly equivalent to standard regression trees and OKBoost reduces to Friedman (2001)'s least-squares gradient boosting.

Image reconstruction (Weston et al., 2002): The goal of this problem is to complete the bottom part of an image representing a handwritten digit from the top part of the image. The dataset contains 1000 16×16 images from the USPS dataset. Inputs are the 128 pixels describing the top half of the image and the output kernel is a gaussian kernel defined on the 128 bottom pixels (with $\sigma = 7.0711$ as in Geurts et al., 2006b).

Network completion (Yamanishi et al., 2005).

³This follows directly from the fact that the i^{th} row (or column) of matrices W^m contains only N_i non zero elements, where N_i is the sample size of the leaf reached by the i^{th} object.

Given a known graph over some vertices described each by some input feature vector, the goal is to predict the graph over some new vertices from their inputs only. This problem is turned into a kernelized output problem by defining a kernel over nodes in the graph such that high kernel values correspond to highly connected vertices. A graph prediction is then computed by thresholding kernel predictions (7) between the inputs of two new vertices. Following (Geurts et al., 2006b), we use the enzyme network of (Yamanishi et al., 2005) that contains 668 vertices described each by 325 (binary and numerical) features and connected by 2782 edges. As an output kernel, we use a diffusion kernel with a diffusion coefficient of 1.

Influence of the parameters. Figure 1 shows the evolution of errors with the number of terms M for different values of ν and J , top on the image reconstruction problem, bottom on the network problem. Similar trends are observed on the Friedman1 problem. These curves were obtained from a learning set/test set split of 200/800 for the first dataset and 334/334 for the second one. In both cases, we measured the average square loss in the output feature space, ie. $E_{x,y}\{|\phi(y) - F_M^\phi(x)|^2\}$ on the learning and test sample (resp. $Err^\phi(LS)$ and $Err^\phi(TS)$). These latter errors can be computed from kernel values by using Eqn (3) and exploiting the special form of predictions in Eqn (5). For the image reconstruction problem, we also measured the average loss of the pre-images obtained with (8), both on the learning and test sample (resp. $Err^Y(LS)$ and $Err^Y(TS)$). For the network completion problem, we estimated the AUC of the ROC curves obtained when varying the threshold for the prediction of all edges in the learning sample ($AUC(LS)$) and of all edges that involve at least one vertex from the test sample ($AUC(TS)$).

Figure 1 shows similar behavior on the two problems. As expected, $Err^\phi(LS)$ decreases as iterations proceed, while the speed of convergence increases with the tree size J and the value of ν . The figure highlights the importance of shrinkage: with a too large ν (leftmost graphs), the training error rapidly decreases but the testing error starts increasing after a few iterations because of overfitting. When ν decreases, the optimum of the test error appears for a larger M and corresponds to a lower value. On both problems, we observed that smaller values of ν lead to better results and help to avoid overfitting as M increases. The behavior of the pre-image error Err^Y and of the AUC are similar to the behavior of kernel errors, even if the algorithm does not try explicitly to optimize them. The optimal values of M according to these criteria indeed match those of the kernel errors.

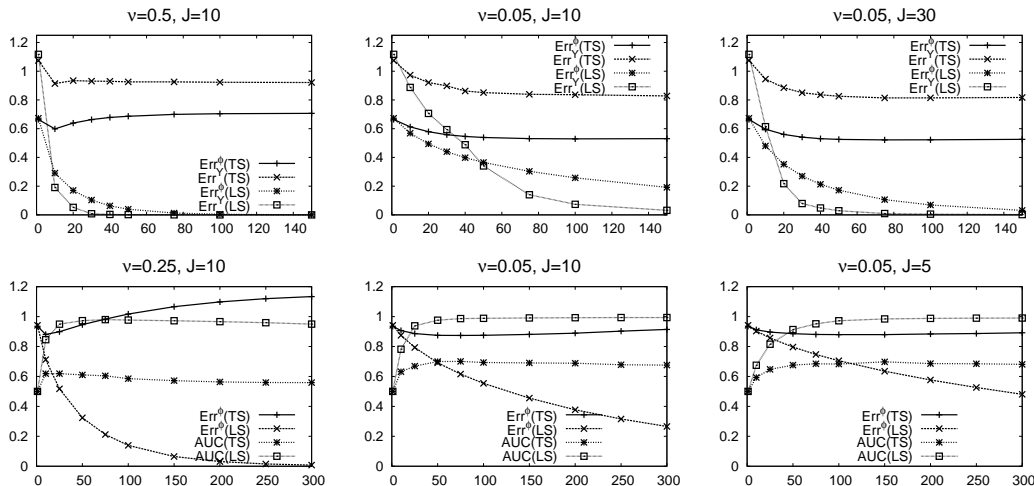


Figure 1. Evolution of different errors with M , top on the image reconstruction problem, bottom on the network completion problem (ν : shrinkage; J : tree complexity)

Table 1. Comparison of OKBoost with other OK3 variants ($\nu = 0.01$; $M = 500$)

Method	Friedman1		Image		Network	
	$Err^\phi(TS)$	$Err^\psi(TS)$	$Err^\phi(TS)$	$Err^\psi(TS)$	$Err^\phi(TS)$	AUC(TS)
OK3 (single trees)	11.203 ± 0.619	11.203 ± 0.619	1.0434 ± 0.0152	1.0399 ± 0.0150	1.6928 ± 0.0472	0.6001 ± 0.0149
OK3+Bagging	5.089 ± 0.394	5.154 ± 0.378	0.5442 ± 0.0019	0.8643 ± 0.0096	0.8824 ± 0.0258	0.7100 ± 0.0205
OK3+ET	5.990 ± 0.561	6.162 ± 0.582	0.5170 ± 0.0024	0.8169 ± 0.0109	0.7955 ± 0.0425	0.7846 ± 0.0172
OK3+OKBoost	3.589 ± 0.277	3.589 ± 0.271	0.5241 ± 0.0021	0.8318 ± 0.0067	0.8730 ± 0.0283	0.7033 ± 0.0203
OK3-et+OKBoost	<u>3.349 ± 0.355</u>	<u>3.365 ± 0.349</u>	<u>0.5093 ± 0.0033</u>	<u>0.8071 ± 0.0114</u>	<u>0.8169 ± 0.0334</u>	<u>0.7811 ± 0.0112</u>

Comparison with other ensemble methods.

Table 1 compares OKBoost with the three OK3 variants explored in (Geurts et al., 2006b), i.e. OK3 as single trees, bagging of OK3, and ensemble of OK3 grown with the randomization of the Extra-trees method (Geurts et al., 2006a). We compare two base learners in OKBoost: standard OK3 and OK3-et (ie. OK3 grown with the randomization of extra-trees). Errors on Friedman1 are estimated from a fixed test sample of size 1000 and are averages over 10 random learning samples of size 300. On the other two problems, errors and AUCs were obtained by 5-fold cross-validation (with learning and test folds of size resp. 200 and 800 on the image problem and standard 5-fold CV on a subset of 334 nodes on the network problem). For both boosting variants, following advices in (Friedman, 2001), we choose a very small value of $\nu = 0.01$ (corresponding to a slow learning rate) and a large number of trees $M = 500$. Better results could be obtained by optimizing these two parameters. The number of splits J was selected in $[1; 40]$ by another round of 5-fold CV internal to the learning folds. We use ensembles of 100 trees for the other ensemble methods (increasing this number does not improve the results) and the default setting for the extra-trees method (Geurts et al., 2006a). Results are reported in Table

1, with the best result in each column underlined.

For each problem, we report the kernel error $Err^\phi(TS)$. On the Network problem, we computed the $AUC(TS)$, and, on the other problems, the pre-image error $Err^\psi(TS)$. This approximation is not required on Friedman1 which exploits a linear kernel but it is reported for comparison purpose. On the first two problems, Err^ϕ and Err^ψ follow very similar trends, i.e., better predictions in \mathcal{H} translates into better pre-images. The difference between Err^ϕ and Err^ψ is much more important on the image problem than on Friedman1. This is not surprising as on the image problem, the output space \mathcal{Y} (all possible bottom images corresponding to a digit) only very sparsely populates the space \mathcal{H} induced by the Gaussian kernel, while on the Friedman1 problem, both spaces coincide.

On all problems, OKBoost with standard OK3 significantly outperforms single trees. It also outperforms bagging on the friedman1 and image problems, but is slightly less good on the network problem. It outperforms Extra-trees significantly on Friedman1, but is less good on the image and network problems. In all cases, the introduction of the extra-trees randomization further improves the results of OKBoost, the most important improvement being observed on the net-

work problem. Globally, OK3-et+OKBoost is the best approach. It is only slightly worse than Extra-trees on the network problem, slightly better on the Image problem, and significantly better on Friedman1⁴. The rather disappointing results of OK3+OKBoost on the network problem suggest that on this problem reducing variance (the main goal of Extra-trees) is more productive than reducing bias (the main goal of boosting). We believe that this is a consequence of the fact that this dataset is very noisy, the prediction of interactions from the inputs being a very difficult task. Results on Friedman1 show nevertheless that reducing bias could be more effective on other problems.

For a comparison of OKBoost with other methods, one may refer to (Geurts et al., 2006b) where results are provided on the image and network problems, respectively with KDE and other network completion algorithms. It turns out that OKBoost is slightly inferior to KDE and competitive with existing graph completion techniques.

4. Conclusion

An extension of gradient boosting to kernelized output spaces has been presented. This framework applies to any base learner making predictions in the form of (input-dependent) weighted averages of sample output values, such as nearest neighbor regression, (local or global) linear regression, or kernel-based regression, provided that it is first reformulated to operate in kernelized output spaces. In particular, in this paper we have applied this approach to output kernelized tree-based regression. The sparse nature of these averagers leads to a significant reduction of computational complexity with respect to the general case. A potential drawback of them is that they are currently not able to exploit kernels defined over the input space.

Experiments on three diverse problems showed that OKBoost substantially improves a single OK3 learner and that its combination with randomization provides competitive results with other tree-based ensemble methods. This behavior is coherent with what has been observed with boosting algorithms in standard regression tasks (Friedman, 2001; Rätsch et al., 2002). The rather important improvement that we have observed when using randomized trees instead of standard ones as base learners, suggests that further progress could be made by focusing on regularization schemes in the context of kernelized output spaces.

⁴The difference is statistically significant in all cases, except on the Network problem with the AUC (according to a paired two-sided t-tests with a confidence level of 95%).

Acknowledgments

PG is a research associate of the FNRS (Belgium). This work has been done while he was a post-doc at IBISC (Evry, France) with support of the CNRS (France). FAB's research has been funded by Genopole (France).

References

- Cortes, C., Mehryar, M., & Weston, J. (2005). A general regression technique for learning transductions. *Proceedings of ICML 2005* (pp. 153–160).
- Freund, Y., & Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55, 119–139.
- Friedman, J. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, 19, 1–67.
- Friedman, J. (2001). Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29, 1189–1232.
- Friedman, J. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38, 367–378.
- Geurts, P., Ernst, D., & Wehenkel, L. (2006a). Extremely randomized trees. *Machine Learning*, 36, 3–42.
- Geurts, P., Wehenkel, L., & d'Alché Buc F. (2006b). Kernelizing the output of tree-based methods. *Proc. of ICML* (pp. 345–352).
- Mason, L., Baxter, J., Bartlett, P., & Frean, M. (2000). Boosting algorithms as gradient descent. *Neural Information Processing Systems* (pp. 512–518).
- Memisevic (2006). *An introduction to structured discriminative learning* (Technical Report). Department of Computer Science, University of Toronto.
- Rätsch, G., Demiriz, A., & Bennett, K. (2002). Sparse regression ensembles in infinite and finite hypothesis spaces. *Machine Learning*, 48, 193–221.
- Szedmak, S., Shawe-Taylor, J., & Parado-Hernandez, E. (2005). *Learning via linear operators: Maximum margin regression* (Technical Report). University of Southampton, UK.
- Taskar, B., Chatalbashev, V., Koller, D., & Guestrin, C. (2005). Learning structured prediction models: A large margin approach. *Proc. of ICML 2005* (pp. 897–904).
- Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *JMLR*, 6, 1453–1484.
- Weston, J., Chapelle, O., Elisseeff, A., Schoelkopf, B., & Vapnik, V. (2002). Kernel dependency estimation. *Advances in Neural Information Processing Systems*, 15.
- Yamanishi, Y., Vert, J.-P., & Kanehisa, M. (2005). Supervised enzyme network inference from the integration of genomic data and chemical information. *Bioinformatics*, 21, i468–i477.