KTBoost: Combined Kernel and Tree Boosting

Fabio Sigrist*
Lucerne University of Applied Sciences and Arts

May 23, 2019

Abstract

In this article, we introduce a novel boosting algorithm called 'KTBoost', which combines **k**ernel boosting and **t**ree boosting. In each boosting iteration, the algorithm adds either a regression tree or reproducing kernel Hilbert space (RKHS) regression function to the ensemble of base learners. Intuitively, the idea is that discontinuous trees and continuous RKHS regression functions complement each other, and that this combination allows for better learning of functions that have parts with varying degrees of regularity such as discontinuities and smooth parts. We empirically show that KTBoost outperforms both tree and kernel boosting in terms of predictive accuracy on a wide array of data sets.

1 Introduction

Boosting algorithms [Freund and Schapire, 1996, Friedman et al., 2000, Mason et al., 2000, Friedman, 2001, Bühlmann and Hothorn, 2007] enjoy large popularity in both applied data analysis and machine learning research due to their high predictive accuracy observed on a wide range of data sets [Chen and Guestrin, 2016]. Boosting additively combines base learners by sequentially minimizing a risk functional. Despite the fact that there is almost no restriction on the type of base learners in the seminal papers of Freund and Schapire [1996] and Freund and Schapire [1997], very little research has been done on combining different types of base learners. To the best of our knowledge, except for one reference [Hothorn et al., 2010], existing boosting algorithms use only one type of functions as base learners. To date, regression trees are the most common choice of base learners, and a lot of effort has been made in recent years to develop tree-based boosting methods that scale to large data [Chen and Guestrin, 2016, Ke et al., 2017, Ponomareva et al., 2017, Prokhorenkova et al., 2018].

In this article, we relax the assumption of using only one type of base learners by combining regression trees [Breiman et al., 1984] and reproducing kernel Hilbert space (RKHS) regression functions [Schölkopf and Smola, 2001, Berlinet and Thomas-Agnan, 2011] as base learners. In short, RKHS regression is a form on non-parametric regression which shows state-of-the-art predictive accuracy for many data sets as it can, for instance, achieve near-optimal test errors [Belkin et al., 2018b,a], and kernel classifiers parallel the

*Email: fabio.sigrist@hslu.ch. Address: Institute of Financial Services Zug IFZ, Lucerne University of Applied Sciences and Arts, Grafenauweg 10, 6302 Zug, Switzerland.

behaviors of deep networks as noted in Zhang et al. [2017]. As there is now growing evidence that base learners do not necessarily need to have low complexity [Wyner et al., 2017], continuous, or smooth, RKHS functions have thus the potential to complement discontinuous trees as base learners.

1.1 Summary of results

We introduce a novel boosting algorithm denoted by 'KTBoost', which combines **k**ernel and **t**ree boosting. In each boosting iteration, the KTBoost algorithm adds either a regression tree or a penalized RKHS regression function, also known as kernel ridge regression [Murphy, 2012], to the ensemble. This is done by first learning both a tree and an RKHS function using one step of functional Newton's method or functional gradient descent, and then selecting the base learner whose addition to the ensemble results in the lowest empirical risk. The KTBoost algorithm thus chooses in each iteration a base learner from two fundamentally different function classes. Functions in an RKHS are continuous and, depending on the kernel function, they also have higher regularity. Trees, on the other hand, are discontinuous functions.

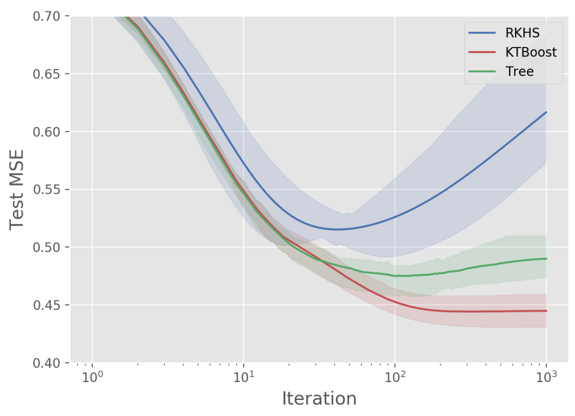

Figure 1: Test mean square error (MSE) versus the number of boosting iteration for KTBoost in comparison with tree and kernel boosting for one data set (wine).

Intuitively, the idea is that the different types of base learners complement each other, and that this combination allows for better learning of functions that exhibit parts with varying degrees of regularity. We demonstrate this effect in a simulation study in Section 4.1. To briefly illustrate that the combination of trees and RKHS functions as base learners can achieve higher predictive accuracy, we report in Figure 1 test mean square errors (MSEs) versus the number of boosting iterations for one data set (wine). The solid lines show average test MSEs over ten random splits into training, validation, and test data sets versus the number of boosting iterations. The confidence bands are obtained after pointwise excluding the largest and smallest MSEs. Tuning parameters of all methods are chosen on the validation data sets. See Section 4 for more details on the data set and the choice of tuning parameters.[1] The figure illustrates how the combination of tree and kernel boosting (KTBoost) results in a lower test MSE compared to both tree and kernel

[1] For better comparison, the shrinkage parameter ν, see Equation (4), is set to a fix value ($\nu = 0.1$) in this example. In the experiments in Section 4.2, the shrinkage parameter is also chosen using cross-validation.

boosting. In our extensive experiments in Section 4.2, we show on a large collection of data sets that the combination of trees and RKHS functions leads to a lower generalization error compared to both only tree and only kernel boosting. Our approach is implemented in the Python package `KTBoost`, which is openly available on the Python Package Index (PyPI) repository.[2]

1.2 Related work

Combining predictions from several models has been successfully applied in many areas of machine learning such as diversity inducing methods, see, e.g., Mendes-Moreira et al. [2012], or multi-view learning, see, e.g., Peng et al. [2018] for a recent example of a boosting application. However, the way boosting combines base learners is different from traditional ensembles consisting of several models trained on potentially different data sets since, for instance, boosting reduces both variance and bias. Very little research has been done on combining different types of base learners in a boosting framework, and, to the best of our knowledge, there is no study which investigates the effect on the predictive accuracy when boosting different types of base learners. A potential explanation for this could be the hypothesis that combining several base learners entails the risk of overfitting.

The `mboost` R package of Hothorn et al. [2010] also allows for combining different base learners which include linear functions, one- and two-dimensional smoothing splines, spatial terms, regression trees, as well as user-defined ones. This approach is different from ours since `mboost` uses a component-wise approach where every base learner typically depends on only a few features, and in each boosting update, the term which minimizes a least squares approximation to the negative gradient of the empirical risk is added to the ensemble. In contrast, in our approach, the tree and the kernel machine depend on all features by default, base learners are learned using Newton's method or gradient descent, and we select the base learner whose addition to the ensemble directly results in the lowest empirical risk.

2 Preliminaries

2.1 Boosting

There exist population as well as sample versions of boosting algorithms. For the sake of brevity, we only consider the latter here. Assume that we have data $\{(x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, i = 1, \dots, n\}$ from a probability distribution $P_{X,Y}$. The goal of boosting is to find a function $F : \mathbb{R}^p \to \mathbb{R}$ in a Hilbert space $\Omega_{\mathcal{S}}$ which allows for predicting y given x. Note that y can be categorical, discrete, continuous, or of mixed type depending on whether the conditional distribution $P_{Y|X}$ is absolutely continuous with respect to the Lebesgue, a counting measure, or a mixture of the two; see, e.g., Sigrist and Hirnschall [2019] for an example of the latter. Depending on the data and the goal of the application, the function $F = (F^k), k = 1, \dots, d$, can also be multivariate. For the sake of notational simplicity, we assume in the following that F is univariate. The extension to the multivariate case is straightforward; see, e.g., Sigrist [2018].

[2]See https://github.com/fabsig/KTBoost for more information.

The goal of boosting is to find a minimizer $F^*(\cdot)$ of the empirical risk functional $R(F)$:

$$F^*(\cdot) = \underset{F(\cdot)\in\Omega_{\mathcal{S}}}{\operatorname{argmin}} R(F) = \underset{F(\cdot)\in\Omega_{\mathcal{S}}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F(x_i)), \tag{1}$$

where $L(Y, F)$ is an appropriately chosen loss function such as the squared error for regression or the logistic regression loss for binary classification. It is assumed that the function space $\Omega_{\mathcal{S}} = span(\mathcal{S})$ is the span of a set of base learners $\mathcal{S} = \{f_j : \mathbb{R}^p \to \mathbb{R}\}$. Boosting then finds $F^*(\cdot)$ in a sequential way by iteratively adding an update f_m to the current estimate F_{m-1}:

$$F_m(x) = F_{m-1}(x) + f_m(x), \quad f_m \in \mathcal{S}, \quad m = 1, \dots, M, \tag{2}$$

such that the empirical risk is minimized

$$f_m = \underset{f\in\mathcal{S}}{\operatorname{argmin}} R\left(F_{m-1} + f\right). \tag{3}$$

Since this usually cannot be found explicitly, one uses an approximate minimizer. Depending on whether gradient or Newton boosting is used, the update f_m is either obtained as the least squares approximation to the negative functional gradient or by applying one step of functional Newton's method which corresponds to minimizing a second order Taylor expansion of the risk functional; see Section 3 or Sigrist [2018] for more information. For increased predictive accuracy [Friedman, 2001], an additional shrinkage parameter $\nu > 0$ is usually added to the update equation:

$$F_m(x) = F_{m-1}(x) + \nu f_m(x). \tag{4}$$

2.2 Reproducing kernel Hilbert space regression

Assume that $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a positive definite kernel function. Then there exists a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ with an inner product $\langle \cdot, \cdot \rangle$ such that (i) the function $K(\cdot, x)$ belongs to $\mathcal{H}$ for all $x \in \mathbb{R}^d$ and (ii) $f(x) = \langle f, K(\cdot, x)\rangle$ for all $f \in \mathcal{H}$. Suppose we are interested in finding the minimizer

$$\underset{f\in\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \|f\|^2_{\mathcal{H}}, \tag{5}$$

where $\lambda \geq 0$ is a regularization parameter. The representer theorem [Schölkopf et al., 2001] then states that there is a unique minimizer of the form $f(\cdot) = \sum_{j=1}^{n} \alpha_j K(x_j, \cdot)$ and (5) can be written as $\|y - K\alpha\|^2 + \lambda \alpha^T K \alpha$, where $y = (y_1, \dots, y_n)^T$, $K \in \mathbb{R}^{n\times n}$ with $K_{ij} = K(x_i, x_j)$, and $\alpha = (\alpha_1, \dots, \alpha_n)^T$. Taking derivatives and equaling them to zero, we find the explicit solution as $\alpha = (K + \lambda I_n)^{-1} y$, where I_n denotes the n-dimensional identity matrix.

There is a close connection between Gaussian process regression and kernel regression. The solution to (5) is the posterior mean conditional on the data of a zero-mean Gaussian process with covariance function K. Further, since $f(x) = k(x)^T (K + \lambda I_n)^{-1} y$ where

$$k(x) = (K(x_1, x), \dots, K(x_n, x))^T, \tag{6}$$

kernel regression can also be interpreted as a two-layer neural network.

2.3 Regression trees

We denote by $\mathcal{T}$ the space which consists of regression trees [Breiman et al., 1984]. Following the notation used in Chen and Guestrin [2016], a regression tree is given by

$$f^T(x) = w_{s(x)},$$

where $s : \mathbb{R}^p \to \{1, \dots, J\}$, $w \in \mathbb{R}^J$, and $J \in \mathbb{N}$ denotes the number of terminal nodes of the tree $f^T(x)$. s determines the structure of the tree, i.e., the partition of the space, and w denotes the leaf values. As in Breiman et al. [1984], we assume that the partition of the space made by s is a binary tree where each cell in the partition is a rectangle of the form $R_j = (l_1, u_1] \times \cdots \times (l_p, u_p] \subset \mathbb{R}^p$ with $-\infty \leq l_m < u_m \leq \infty$ and $s(x) = j$ if $x \in R_j$.

3 Combined kernel and tree boosting

Let $R^2(F_{m-1} + f)$ denote the functional, which is proportional to a second order Taylor approximation of the empirical risk in (1) at the current estimate F_{m-1}:

$$R^2(F_{m-1} + f) = \sum_{i=1}^{n} g_{m,i} f(x_i) + \frac{1}{2} h_{m,i} f(x_i)^2, \tag{7}$$

where $g_{m,i}$ and $h_{m,i}$ are the gradient and the Hessian

$$g_{m,i} = \frac{\partial}{\partial F} L(y_i, F)\Big|_{F=F_{m-1}(x_i)} \quad \text{and} \quad h_{m,i} = \frac{\partial^2}{\partial F^2} L(y_i, F)\Big|_{F=F_{m-1}(x_i)}. \tag{8}$$

The KTBoost algorithm then works as follows. In each boosting iteration, a candidate tree $f_m^T(x)$ and RKHS function $f_m^K(x)$ are found as minimizers of the second order Taylor approximation $R^2(F_{m-1}+f)$. This corresponds to applying one step of Newton's method. It can be shown that candidate trees $f_m^T(x)$ can be found as weighted least squares minimizers; see, e.g., Chen and Guestrin [2016] or Sigrist [2018]. Further, the candidate penalized RKHS regression functions $f_m^K(x)$ can be found as shown in Proposition 3.1. The KTBoost algorithm then selects either the tree or the RKHS function such that the addition of the base learner to the ensemble according to Equation (4) results in the lowest risk. Algorithm 1 summarizes this.

Proposition 3.1. *The optimal kernel ridge regression solution $f_m^K(x)$ in the Newton update step of boosting is given by $f_m^K(x) = k(x)^T \alpha_m$, where $k(x)$ is defined in* (6) *and*

$$\alpha_m = D_m \left(D_m K D_m + \lambda I_n\right)^{-1} D_m y_m, \tag{9}$$

where $D_m = diag\left(\sqrt{h_{m,i}}\right)$, $y_m = (-g_{m,1}/h_{m,1}, \dots, -g_{m,n}/h_{m,n})^T$, and I_n is the identity matrix of dimension n.

Proof. We have

$$\begin{aligned}\min_{f\in\mathcal{H}} \sum_{i=1}^{n} g_{m,i} f(x_i) + \frac{1}{2} h_{m,i} f(x_i)^2 + \lambda \|f\|_{\mathcal{H}}^2 &= \min_{f\in\mathcal{H}} \sum_{i=1}^{n} h_{m,i} \left(-\frac{g_{m,i}}{h_{m,i}} - f(x_i)\right)^2 + \lambda \|f\|_{\mathcal{H}}^2 \\ &= \min_{\alpha} \|D_m y_m - D_m K \alpha\|^2 + \lambda \alpha^T K \alpha.\end{aligned}$$

If we take derivatives with respect to α, equal them to zero, and solve for α, we find that $\alpha_m = \left(K D_m D_m K + \lambda K\right)^{-1} K D_m y_m = D_m \left(D_m K D_m + \lambda I_n\right)^{-1} D_m y_m$. □

Algorithm 1: KTBoost

```
Initialize $F_0(x) = \text{argmin}_{c \in \mathbb{R}^d} R(c)$.
for $m = 1$ to $M$ do
  Compute the gradient $g_{m,i}$ and Hessian $h_{m,i}$ as defined in (8)
  Find the candidate regression tree $f_m^T(x)$ and RKHS function $f_m^K(x)$

      $f_m^T(x) = \text{argmin}_{f \in \mathcal{T}} R^2(F_{m-1} + f)$

      $f_m^K(x) = \text{argmin}_{f \in \mathcal{H}} R^2(F_{m-1} + f) + \lambda \|f\|_{\mathcal{H}}^2$

  where the approximate risk $R^2(F_{m-1} + f)$ is defined in (7)
  if $R\left(F_{m-1} + \nu f_m^T(x)\right) \leq R\left(F_{m-1} + \nu f_m^K(x)\right)$ then
    $f_m(x) = f_m^T(x)$
  else
    $f_m(x) = f_m^K(x)$
  end if
  Update $F_m(x) = F_{m-1}(x) + \nu f_m(x)$
end for
```

In the above formulation, KTBoost uses functional Newton's method to find the updates f_m. If either the loss function is not twice differentiable in its second argument or the second derivative is zero or constant on a non-null set of the support of X, one can alternatively use gradient boosting. This has the advantage that it is computationally less expensive than Newton boosting since, in contrast to (9), the kernel matrix does not depend on the iteration number m; see Section 3.1 for more details. Note that the gradient boosting version of KTBoost is simply obtained as a special case of the Algorithm 1 by setting $h_{m,i} = 1$. Further, note that for the RKHS boosting part, the update equation $F_m(x) = F_{m-1}(x) + \nu f_m(x)$ can be replaced by simply updating the coefficients α_m.

3.1 Reducing computational costs for large data

The computationally expensive part for finding the kernel regression updates is the factorization of the kernel matrix which scales with $O(n^3)$. There are several approaches that allow for computational efficiency in the large data case. Examples of this include low rank approximations based on, e.g., the Nyström method [Williams and Seeger, 2001] and extensions of it such as divide-and-conquer kernel ridge regression [Zhang et al., 2013, 2015], and early stopping of iterative optimization methods [Yao et al., 2007, Blanchard and Krämer, 2010, Raskutti et al., 2014, Ma and Belkin, 2017]. Another approach adopted mainly in spatial statistics is to use a kernel function that has a compact support [Gneiting, 2002, Bevilacqua et al., 2018]. The resulting kernel matrix K is then sparse and can thus be efficiently inverted or factorized. Further, if gradient descent is used instead of Newton's method, the RKHS function $f_m^K(x)$ can be found efficiently by observing that, in contrast to (9), the kernel matrix $K + \lambda I_n$ does not depend on the iteration number m. I.e., it inverse or a Cholesky factor of it needs to be calculated only once. Concerning the regression trees, finding the splits when growing the trees is the computationally demanding part. There are several approaches in the literature on how this can be done efficiently for large data; see, e.g., Chen and Guestrin [2016] or Ke et al. [2017]. Further, the two learners can be learned in parallel.

In our empirical analysis, we use the Nyström method for dealing with large data sets. The Nyström method approximates the kernel $K(\cdot,\cdot)$ by first choosing a set of l so-called Nyström samples $x_1^*, \dots, x_l^*$. Often these are obtained by sampling uniformly from the data. Denoting the kernel matrix that corresponds to these points as K^*, the Nyström method then approximates the kernel $K(\cdot,\cdot)$ as $K(x,y) \approx k^*(x)^T K^{*-1} k^*(y)$, where $k^*(x) = (K(x,x_1^*), \dots, K(x,x_l^*))^T$. In particular, the reduced-rank Nyström approximation to the full kernel matrix K is given by $K \approx K K^{*-1} K^T$.

4 Experimental results

4.1 Simulation study

We first conduct a small simulation study to illustrate that the combination of discontinuous trees and continuous kernel machines can indeed better learn functions with both discontinuous and smooth parts. We consider random functions $F : [0,1] \to \mathbb{R}$ with five random jumps in $[0, 0.5]$:

$$F(x) = \sum_{i=1}^{5} g_i \mathbf{1}_{(t_i,1]}(x) + \sin(8\pi x), \quad t_i \overset{\text{iid}}{\sim} \text{Unif}(0, 0.5), \quad g_i \overset{\text{iid}}{\sim} \text{Unif}(0, 5)$$

and data according to $y_i = F(x_i) + N(0, 0.25^2)$, $x_i \overset{\text{iid}}{\sim} \text{Unif}(0,1)$, $i = 1, \dots, 1000$. In Figure 2 on the left-hand side, an example of such a function and corresponding data is shown. We simulate 1000 times such random functions as well as training, validation, and test data of size $n = 1000$. For each simulation run, learning is done on the training data. The number of boosting iterations is chosen on the validation data with the maximum number of boosting iterations being $M = 1000$. We use a learning rate of $\nu = 0.1$ as this is a reasonable default value [Bühlmann and Hothorn, 2007] and trees of depth $= 1$ as there are no interactions. In Figure 2 on the right-hand side, we show the pointwise test mean square error (MSE) for tree and kernel boosting as well as the combined KTBoost algorithm. We observe that tree boosting performs better than kernel boosting in the area where the discontinuities are located and, conversely, kernel boosting outperforms tree boosting on the smooth part. The figure also clearly shows that KTBoost outperforms both tree and kernel boosting as it achieves the MSE of tree boosting on the interval with jumps and the MSE of kernel boosting on the smooth part.

4.2 Real-world data

In the following, we compare the KTBoost algorithm with tree and kernel boosting using the following Delve, Keel, Kaggle, and UCI data sets: abalone, ailerons, bank8FM, elevators, energy, housing, liberty, NavalT, parkinsons, puma32h, sarcos, wine, adult, cancer, ijcnn, ionosphere, sonar, car, epileptic, glass, and satimage. Detailed information on the number of samples and features can be found in the supplementary material. We consider both regression as well as binary and multiclass classification data sets. Further, we include data sets of different sizes in order to investigate the performance on both smaller and larger data sets, as small- to moderately-sized data sets continue to be widely used in applied data science despite the recent focus on very large data sets in machine learning research. We

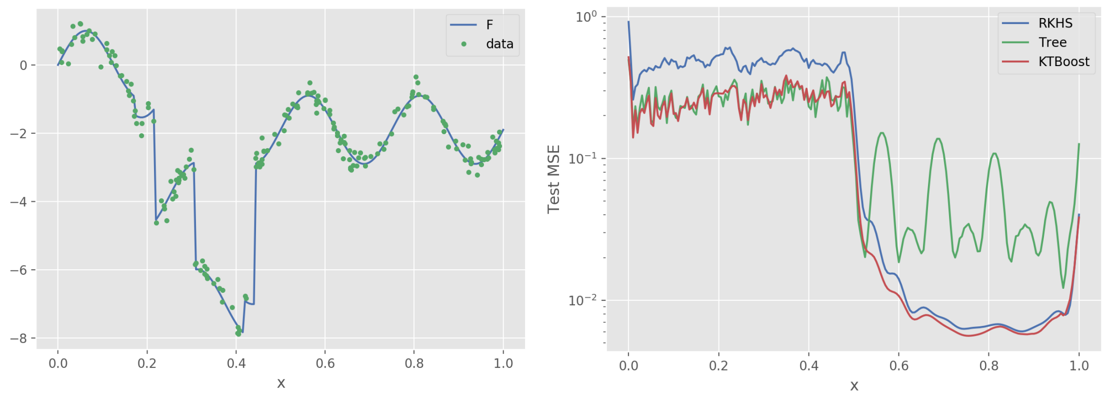

Figure 2: An example of a random function with five random jumps in $[0, 0.5]$ and corresponding observed data (left plot) and pointwise mean square error (MSE) for tree and kernel boosting as well as the combined KTBoost algorithm (right plot).

use the squared loss for regression, the logistic regression loss for binary classification, and the cross-entropy loss with the softmax function for multiclass classification. Further, for the RKHS ridge regression, we use a Gaussian kernel $K(x_1, x_2) = \exp\left(-\|x_1 - x_2\|^2/\rho^2\right)$.

For the regression data sets, we use gradient boosting, and for the classification data sets, we use boosting with Newton updates since this can result in more accurate predictions [Sigrist, 2018]. For some classification data sets (adult, ijcnn, epileptic, and satimage), Newton boosting is computationally infeasible on a standard single CPU computer with the current implementation of KTBoost, despite the use of the Nyström method with a reasonable number of Nyström samples, say 1000, since the weighted kernel matrix in Equation (9) needs to be factorized in every iteration. We thus also use gradient boosting for these data sets. Technically, it would be possible for these cases to learn the trees using Newton's method, or using the hybrid gradient-Newton boosting version of Friedman [2001], but this would result in an unfair comparison that is biased in favor of the base learner which is learned with the better optimization method. For the larger data sets (liberty, sarcos, adult, ijcnn), we use the Nyström method described in Section 3.1. Specifically, we use $l = 1000$ Nyström samples, which are uniformly sampled from the training data. In general, the larger the number of Nyström samples, the lower the approximation error but the higher the computational costs. Williams and Seeger [2001] report good results with $l \approx 1000$ for several data sets. All calculations are done with the Python package `KTBoost` on a standard laptop with a 2.9 GHz quad-core processor and 16GB of RAM.

All data sets are randomly split into three parts of equal size to obtain training, validation and test sets. Learning is done on the training data, tuning parameters are chosen on the validation data, and model comparison is done on the test data. All input features are standardized using the training data to have approximately mean zero and variance one. In order to quantify variability in the results, we use ten different random splits of the data into training, validation and test sets.

Concerning tuning parameters, we select the number of boosting iterations M from $\{1, 2, \ldots, 1000\}$, the learning rate ν from $\{1, 10^{-1}, 10^{-2}, 10^{-3}\}$, the maximal depth of the trees from $\{1, 5, 10\}$, and the kernel ridge regularization parameter λ from $\{1, 10\}$. Further, the kernel range parameter ρ is chosen using k-nearest neighbors distances as described in the following. We first calculate the average distance of all k-nearest neighbors in the training data, where k is a tuning parameter selected from $\{5, 50, 500, 5000, n-1\}$ and n is the size of the training data. We then choose ρ such that the kernel function has decayed to a value of 0.01 at this average k-nearest neighbors distance. This is motivated by the fact that for a corresponding Gaussian process with such a covariance function, the correlation has decayed to a level of 1% at this k-nearest neighbor distance. If the training data contains less than 5000 (or 500) samples, we use $n-1$ as the maximal number for the k-nearest neighbors. In addition, we include ρ which equals the average $(n-1)$-nearest neighbor distance. The latter choice is done in order to also include a range which results in a kernel that decays slowly over the entire space. For the large data sets where the Nyström method is used, we calculate the average k-nearest neighbors distance based on the Nyström samples. I.e., in this case, the maximal k equals $l-1$.

Table 1: Comparison of KTBoost with tree and kernel boosting using test mean square error (regression) and test error rate (classification). In parentheses are standard deviations. For comparing KTBoost with tree and kernel boosting, p-values are obtained using t-tests by taking into account that the results of the different methods are dependent for the same sample split. The symbols ‘$^{+}$’ and ‘$^{-}$’ denote the number of significantly better and worse results of KTBoost at the 5% level.

Data	KTBoost	Tree	p-val	Kernel	p-val
abalone	4.65^{+} (0.248)	5.07 (0.261)	3.73e-07	**4.64** (0.255)	0.557
ailerons	**2.64e-08**$^{+}$ (6.19e-10)	8.11e-08 (2.39e-09)	1.23e-13	2.64e-08 (6.19e-10)	1
bank8FM	**0.000915**$^{++}$ (4.02e-05)	0.000945 (2.47e-05)	0.0221	0.000945 (5.83e-05)	0.0351
elevators	**4.83e-06**$^{++}$ (2.9e-07)	5.66e-06 (1.44e-07)	5.45e-06	5.18e-06 (3.89e-07)	0.000153
energy	**0.282**$^{+}$ (0.0372)	0.335 (0.093)	0.0657	1.3 (0.377)	1.32e-05
housing	**12.7** (3.19)	15.1 (3.23)	0.0821	13.6 (2.51)	0.253
liberty	**14.5**$^{+}$ (0.323)	14.5 (0.314)	0.715	15.2 (0.345)	2.36e-07
NavalT	**6.51e-09**$^{+}$ (1.15e-09)	1.15e-06 (1.58e-07)	2.72e-09	6.51e-09 (1.15e-09)	1
parkinsons	**73.3**$^{+}$ (1.98)	81.1 (2.44)	1.51e-06	73.3 (1.91)	0.145
puma32h	**6.5e-05**$^{+}$ (2.27e-06)	6.51e-05 (2.13e-06)	0.375	0.000695 (2.2e-05)	7.33e-15
sarcos	**7.99**$^{++}$ (0.206)	9.6 (0.207)	4.69e-09	17.8 (0.586)	8.99e-13
wine	**0.444**$^{++}$ (0.012)	0.471 (0.0169)	6.78e-05	0.506 (0.0106)	3.78e-08
adult	0.128^{+} (0.00295)	**0.128** (0.00313)	0.589	0.163 (0.00512)	2.95e-10
cancer	0.0362 (0.00744)	0.0415 (0.0153)	0.2	**0.0358** (0.0107)	0.891
ijcnn	**0.0122**$^{+}$ (0.000685)	0.0123 (0.000702)	0.258	0.0387 (0.00516)	6.14e-08
ionosphere	**0.0872**$^{++}$ (0.017)	0.103 (0.0226)	0.0434	0.107 (0.0239)	0.0375
sonar	0.194^{+} (0.0394)	0.223 (0.05)	0.00847	**0.193** (0.0491)	0.853
car	**0.0399** (0.00505)	0.0411 (0.00685)	0.591	0.041 (0.00624)	0.647
epileptic	**0.354**$^{++}$ (0.00612)	0.373 (0.00614)	2.86e-06	0.442 (0.0265)	2.92e-06
glass	**0.308** (0.0711)	0.315 (0.0589)	0.66	0.344 (0.0581)	0.19
satimage	**0.089**$^{+}$ (0.00452)	0.112 (0.00504)	4.03e-07	0.0903 (0.00417)	0.185

The results are shown in Table 1. For the regression data sets, we show the average test mean square error (MSE) over the different sample splits, and for the classification data sets, we calculate the average test error rate. The numbers in parentheses are standard deviations over the different sample splits. p-values are obtained from t-tests comparing KTBoost with tree and kernel boosting. All tests are done using a regression model with a random effect at the sample split level to account for correlation among the results for different sample splits. We find that KTBoost achieves higher predictive accuracy than both tree and kernel boosting for the large majority of data sets. In particular, KTBoost performs either significantly better than both tree and kernel boosting or the difference in predictive accuracy is not significant. Specifically, KTBoost achieves higher predictive accuracy than both tree and kernel boosting for seventeen out of twenty-one data sets, and it is significantly better than both tree-only and kernel-only boosting at the 5% significance level for six data sets and significantly better than either tree or kernel boosting for further eleven data sets. In only four cases either kernel or tree boosting performs marginally better than KTBoost, and these difference are clearly not significant with p-values above 0.5.

Note that we do not report the optimal tuning parameters since this is infeasible for all combinations of data sets and sample splits, and aggregate values are not meaningful since different tuning parameters often compensate each other in a non-linear way (e.g., number of iterations, learning rate, and tree depth or kernel regularization λ). Unfortunately, it is also difficult to concisely summarize the composition of the ensembles in terms of different base learners as a base learner that is added in an earlier boosting stage is more important than one that is added in a later stage [Bühlmann and Yu, 2003], and the properties of the base learners also depend on the chosen tuning parameters. Further, we note that one can also consider additional tuning parameters. For trees, this includes the minimal number of samples per leaf, row and column sub-sampling, and penalization of leave values, and for the kernel regression, this includes the smoothness of the kernel function, or, in general, the class of kernel functions. One can also use different learning rates for the two types of base learners. Due to limits on computational costs, we have not considered all possible combinations of tuning parameters. However, it is likely that a potential increase in predictive performance in either tree or kernel boosting will also result in an increase in accuracy of the combined KTBoost algorithm. We also note that in our experimental setup, the tuning parameter grid for the KTBoost algorithm is larger compared to the tree and kernel boosting cases. This seems inevitable in order to allow for the fairest possible comparison, though. Restricting one type of tuning parameters for the combined version but not for the single base learner case seems to be no alternative. Somewhat alleviating this concern is the fact that, in the above simulation study, we also find outperformance when not choosing tuning parameters using cross-validation, and on the downside, a larger tuning parameter grid might potentially also lead to overfitting. Finally, we remark that we have also considered to compare the risk of the un-damped base learners $\left(R\left(F_{m-1}+f_m^T(x)\right) \leq R\left(F_{m-1}+f_m^K(x)\right)\right)$ when selecting the base learners that is added to the ensemble, and we obtain very similar results (see supplementary material).

5 Conclusions

We have introduced a novel boosting algorithm, which combines trees and RKHS functions as base learners. Intuitively, the idea is that discontinuous trees and continuous RKHS functions complement each other since trees are better suited for learning rougher parts of functions and RKHS regression functions can better learn smoother parts of functions. We have compared the predictive accuracy of KTBoost with tree and kernel boosting in an extensive empirical study and find that KTBoost results in higher predictive accuracy compared to tree and kernel boosting for the large majority of data sets. In particular, KTBoost either achieves significantly higher predictive accuracy or no significant differences are observed in our empirical evaluation.

Future research can be done in several directions. It would be interesting to investigate to which extent other base learners such as neural networks [Huang et al., 2018, Nitanda and Suzuki, 2018] can be used in addition to trees and kernel regression functions. Further, theoretical results such as learning rates or bounds on the risk could help to shed further insights on why the combination of trees and kernel machines leads to increased predictive accuracy. Finally, it would be interesting to compare the KTBoost algorithm on very large data sets. This requires that the algorithm be implemented in a distributed manner. Such an implementation can be done using an existing distributed tree algorithm as in Chen and Guestrin [2016], Ke et al. [2017], or Prokhorenkova et al. [2018] as well as a RKHS regression algorithm and implementation that scales to very large data. Concerning the latter, several potential strategies on how this can be done are briefly outlined in Section 3.1.

References

M. Belkin, D. J. Hsu, and P. Mitra. Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2306–2317, 2018a.

M. Belkin, S. Ma, and S. Mandal. To understand deep learning we need to understand kernel learning. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 541–549, 2018b.

A. Berlinet and C. Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics.* Springer Science & Business Media, 2011.

M. Bevilacqua, T. Faouzi, R. Furrer, and E. Porcu. Estimation and prediction using generalized wendland covariance functions under fixed domain asymptotics. *The Annals of Applied Statistics (accepted)*, 2018.

G. Blanchard and N. Krämer. Optimal learning rates for kernel conjugate gradient regression. In *Advances in Neural Information Processing Systems*, pages 226–234, 2010.

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees.* CRC press, 1984.

P. Bühlmann and T. Hothorn. Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science*, pages 477–505, 2007.

P. Bühlmann and B. Yu. Boosting with the l 2 loss: regression and classification. *Journal of the American Statistical Association*, 98(462):324–339, 2003.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156. Bari, Italy, 1996.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

J. Friedman, T. Hastie, R. Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28 (2):337–407, 2000.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

T. Gneiting. Compactly supported correlation functions. *Journal of Multivariate Analysis*, 83(2):493–508, 2002.

T. Hothorn, P. Bühlmann, T. Kneib, M. Schmid, and B. Hofner. Model-based boosting 2.0. *Journal of Machine Learning Research*, 11(Aug):2109–2113, 2010.

F. Huang, J. Ash, J. Langford, and R. Schapire. Learning deep resnet blocks sequentially using boosting theory. In *ICML*, volume 80, pages 2058–2067, 2018.

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3149–3157, 2017.

S. Ma and M. Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. In *Advances in Neural Information Processing Systems*, pages 3778–3787, 2017.

L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In *Advances in neural information processing systems*, pages 512–518, 2000.

J. Mendes-Moreira, C. Soares, A. M. Jorge, and J. F. D. Sousa. Ensemble approaches for regression: A survey. *Acm computing surveys (csur)*, 45(1):10, 2012.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

A. Nitanda and T. Suzuki. Functional gradient boosting based on residual network perception. In *ICML*, volume 80, pages 3819–3828, 2018.

J. Peng, A. J. Aved, G. Seetharaman, and K. Palaniappan. Multiview boosting with information propagation for classification. *IEEE transactions on neural networks and learning systems*, 29(3):657–669, 2018.

N. Ponomareva, S. Radpour, G. Hendry, S. Haykal, T. Colthurst, P. Mitrichev, and A. Grushetsky. Tf boosted trees: A scalable tensorflow based framework for gradient boosting. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 423–427. Springer, 2017.

L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems 31*, pages 6638–6648. Curran Associates, Inc., 2018.

G. Raskutti, M. J. Wainwright, and B. Yu. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *The Journal of Machine Learning Research*, 15(1):335–366, 2014.

B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

F. Sigrist. Gradient and newton boosting for classification and regression. *arXiv preprint arXiv:1808.03064*, 2018.

F. Sigrist and C. Hirnschall. Grabit: Gradient tree-boosted tobit models for default prediction. *Journal of Banking & Finance*, 102:177 – 192, 2019.

C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pages 682–688, 2001.

A. J. Wyner, M. Olson, J. Bleich, and D. Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *Journal of Machine Learning Research*, 18 (48):1–33, 2017.

Y. Yao, L. Rosasco, and A. Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

Y. Zhang, J. Duchi, and M. Wainwright. Divide and conquer kernel ridge regression. In *Conference on Learning Theory*, pages 592–617, 2013.

Y. Zhang, J. Duchi, and M. Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *The Journal of Machine Learning Research*, 16(1):3299–3340, 2015.

Supplementary material

Summary of data sets

Table 2: Summary of data sets.

Data	# classes	Nb. samples	Nb. features
abalone	regression	4177	10
ailerons	regression	13750	40
bank8FM	regression	8192	8
elevators	regression	16599	18
energy	regression	768	8
housing	regression	506	13
liberty	regression	50999	117
NavalT	regression	11934	16
parkinsons	regression	5875	16
puma32h	regression	8192	32
sarcos	regression	48933	21
wine	regression	4898	11
adult	2	48842	108
cancer	2	699	9
ijcnn	2	141691	22
ionosphere	2	351	34
sonar	2	208	60
car	4	1728	21
epileptic	5	11500	178
glass	7	214	9
satimage	6	6438	36

Results when the shrinkage parameter is not used for choosing the base learner

Table 3: Comparison of KTBoost with tree and kernel boosting using test mean square error (regression) and test error rate (classification). In parentheses are standard deviations. For comparing KTBoost with tree and kernel boosting, p-values are obtained using t-tests by taking into account that the results of the different methods are dependent for the same sample split. The symbols '$^{+}$' and '$^{-}$' denote the number of significantly better and worse results of KTBoost at the 5% level.

Data	KTBoost	Tree	p-val	Kernel	p-val
abalone	**4.63**$^{+}$ (0.246)	5.07 (0.261)	2.25e-07	4.64 (0.255)	0.672
ailerons	2.64e-08^{+} (6.19e-10)	8.11e-08 (2.39e-09)	1.23e-13	**2.64e-08** (6.19e-10)	0.383
bank8FM	**0.000908**$^{++}$ (3.89e-05)	0.000945 (2.47e-05)	0.005	0.000945 (5.83e-05)	0.0072
elevators	**4.71e-06**$^{++}$ (2.16e-07)	5.66e-06 (1.44e-07)	1.84e-07	5.18e-06 (3.89e-07)	3.54e-05
energy	**0.28**$^{+}$ (0.039)	0.335 (0.093)	0.054	1.3 (0.377)	1.3e-05
housing	**12.8** (3.42)	15.1 (3.23)	0.108	13.6 (2.51)	0.297
liberty	**14.5**$^{+}$ (0.299)	14.5 (0.314)	0.932	15.2 (0.345)	2.13e-07
NavalT	6.51e-09^{+} (1.15e-09)	1.15e-06 (1.58e-07)	2.72e-09	**6.51e-09** (1.15e-09)	0.457
parkinsons	73.4^{+} (2.02)	81.1 (2.44)	2.46e-06	**73.3** (1.91)	0.526
puma32h	6.51e-05^{+} (2.27e-06)	**6.51e-05** (2.13e-06)	0.748	0.000695 (2.2e-05)	7.55e-15
sarcos	**7.71**$^{++}$ (0.224)	9.6 (0.207)	6.51e-10	17.8 (0.586)	1.26e-12
wine	**0.463**$^{+}$ (0.0124)	0.471 (0.0169)	0.129	0.506 (0.0106)	1.82e-06
adult	**0.128**$^{+}$ (0.00304)	0.128 (0.00313)	0.779	0.163 (0.00512)	2.71e-10
cancer	**0.0349** (0.00873)	0.0415 (0.0153)	0.257	0.0358 (0.0107)	0.591
ijcnn	**0.0122**$^{+}$ (0.000567)	0.0123 (0.000702)	0.534	0.0387 (0.00516)	5.99e-08
ionosphere	**0.0855**$^{++}$ (0.018)	0.103 (0.0226)	0.0289	0.107 (0.0239)	0.0282
sonar	0.194^{+} (0.0394)	0.223 (0.05)	0.00847	**0.193** (0.0491)	0.853
car	**0.0405** (0.0059)	0.0411 (0.00685)	0.78	0.041 (0.00624)	0.78
epileptic	**0.351**$^{++}$ (0.00543)	0.373 (0.00614)	2.55e-07	0.442 (0.0265)	2.24e-06
glass	0.322 (0.0702)	**0.315** (0.0589)	0.742	0.344 (0.0581)	0.357
satimage	**0.0892**$^{+}$ (0.00399)	0.112 (0.00504)	8.59e-08	0.0903 (0.00417)	0.192