Boosting Kernel Models for Regression

Ping Sun and Xin Yao
School of Computer Science
University of Birmingham
{p.sun,x.yao}@cs.bham.ac.uk

Abstract

This paper proposes a general boosting framework for combining multiple kernel models in the context of both classification and regression problems. Our main approach is built on the idea of gradient boosting together with a new regularization scheme and aims at reducing the cubic complexity of training kernel models. We focus mainly on using the proposed boosting framework to combine kernel ridge regression (KRR) models for regression tasks. Numerical experiments on four large-scale data sets have shown that boosting multiple small KRR models is superior to training a single large KRR model on both improving generalization performance and reducing computational requirements.

1 Introduction

The emergence of kernel-based methods originated from the success of support vector machines (SVM) in pattern recognition [1]. Afterwards, a number of powerful kernel methods, e.g., kernel ridge regression (KRR) [2], kernel principle component analysis (KPCA) [14], kernel fisher discriminant (KFD) [10] and kernel logistic regression (KLR) [23] were proposed and have shown practical relevance for both supervised and unsupervised problems. Here we just consider the supervised problem, which can be formalized as the problem of inferring a function $y = f(x)$ from a given traing dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$. When $y \in \mathbb{R}$ is continuous, the problem is regression, whereas in classification problems, y is categorical (e.g., binary, $y \in \{-1, 1\}$).

Generally, kernel-based methods [15, 23] can be interpreted as the variational problem of finding the function f that minimizes the functional

$$\min_{f \in \mathcal{H}} H[f] = \frac{1}{N} \sum_{i=1}^{N} V(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2 \qquad (1)$$

where $V(\cdot, \cdot)$ is a *loss function*, $\mathcal{H}$ is the Reproducing Kernel Hilbert Spaces (RKHS) generated by the kernel $k(\cdot, \cdot)$ and λ is a parameter that trades off the two terms. The first term assesses the quality of the prediction $f(x_i)$ for the observed target y_i. The second one is called the *regularization* term and represents smoothness assumptions on f. The solution to the problem (1) was given by the well-known *representer theorem* [13] which shows that each minimizer $f \in \mathcal{H}$ of $H[f]$ has the form of

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x). \qquad (2)$$

Substituting $f(x)$ for (1) and using the property of RKHS $\langle k(x_i, \cdot), k(x_j, \cdot) \rangle = k(x_i, x_j)$, we obtain

$$\min_{\alpha} H(\alpha) = \frac{1}{N} \sum_{i=1}^{N} V(y_i, (K\alpha)_i) + \frac{\lambda}{2} \alpha^\top K \alpha, \qquad (3)$$

where $\alpha = [\alpha_1, ..., \alpha_N]^\top$ is the weight parameter, K is the so-called *kernel matrix* of size $N \times N$ and $(K)_{ij} = k(x_i, x_j), i, j = 1, ...N$. By defining different types of loss functions, the problem (3) corresponds to different kernel models. For example, *soft margin* loss function $V(y_i, f(x_i)) = \max\{0, 1 - y_i f(x_i)\}^2$ leads to the primal formulation of SVM, *logistic* loss function results in KLR and KRR can be obtained by replacing $V(\cdot, \cdot)$ with a simple *squared* loss [1].

It is noted that different kernel models require different techniques to find the optimal coefficients α. For training KRR and KLR models, it requires solving a linear system of equations once or repeatedly. And the dual formulation of SVM is a quadratic programming (QP) problem with linear constraints. But a common drawback of all these training algorithms is that the computational cost tends to scale with the cube of the number of training examples (i.e., $\mathcal{O}(N^3)$) and the memory requirements grow as the square, which is impractical for large-scale datasets. To overcome these limitations, numerous speed-up training algorithms

[1] Note that KRR was known as *Regularization Networks* (RN) in the theory of regularization [5]. Additionally, KRR is also equivalent to the MAP estimate of *Gaussian Process Regression* (GPR) [9, 19].

with widely different motivations for each of them (especially for SVM) have been studied extensively in the literature, for example [11, 21, 3, 4, 8, 23, 9].

One general approach to scale up these kernel models to large-scale datasets is that of *divide-and-conquer* strategy. The basic idea is that the dataset is divided up into M small subsets, i.e., $\{\mathcal{D}^j, j = 1, ..., M\}$, and M sub-models are derived from the individual sets. The final prediction is generated by an ensemble model $\bar{f}(x)$ which combine predictions of all M individual models, that is

$$\bar{f}(x) = \sum_{j=1}^{M} c_j f_j(x), \tag{4}$$

where $f_j(x)$ is the j-th sub-model trained on $\mathcal{D}^j$ and c_j is the corresponding weight coefficient. Since the training of individual model on smaller dataset can be very fast, the overall learning cost of M sub-models together with the combining effort are still below the cost of training a single model on the whole dataset. The difficulty of this approach is how to nicely combine the individual results if we assume that small training subsets are randomly generated from the whole dataset.

The methods for weighting individual sub-models appeared in the kernel learning community can be categorized into two classes. The first one is *Mixture-of-Experts*-like approach which dynamically adjust the combining coefficients involved in the composite model. For example, in [4], the authors proposed a parallel mixture of SVMs which combine their individual outputs using a trained MLP neural networks. Another representative example is Bayesian Committee Machine (BCM) [20] which was proposed to scale up the training of Bayesian kernel models. It weights each sub-model by the inverse covariance of its prediction. A good performance of BCM requires that M subsets be pairwisely independent.

The second class of approaches is linearly weighting schemes which do not change the combining coefficients in the phase of testing. The naive example is simple averaging (AV) but it seems not suitable for combining stable kernel models to our best experience. A better idea [22] is to find an optimal combination based on optimizing an objective function under the constraints $\sum_j c_j = 1$ and $c_j \geq 0, j = 1, ..., M$. Pavlov et. al. [11] extended the *Adaboost* technique to combine SVMs for classification problems. They claimed that making an ensemble of few SVMs, each trained on subsets between $2 - 4\%$ of the original data set, results into algorithms whose performance is comparable to a standard SVM trained on the whole dataset.

In this paper, we propose a general framework to nicely combine multiple kernel models for both classification and regression problems. Our work is based on the ideas of *gradient boosting* attached by *a new regularization scheme*. In particular, we demonstrate how to combine multiple KRR models with the proposed framework. Compared to previous combining methods mentioned above, our work is simple to use, does not require any critical parameters and can be applied to boost different kinds of kernel-based models. More important, our boosting approach could achieve better prediction results than a single kernel model on the whole training data, also with significantly lower computational requirements.

2 Boosting Kernel Models

2.1 Gradient Boosting

Gradient boosting [6] is a general framework to strengthen the weak base learners. In our case, the 'weakness' comes from learning kernel models on a subset of a given training set. Following the pioneering work by Friedman [7], the boosting procedure can be generally viewed as a forward stagewise search for a good additive model. This is done by searching, at each iteration, for the base learner which gives the largest reduction in the loss denoted by $L(y, \bar{f})$ [2] , and changing its coefficients accordingly, where $\bar{f}$ denotes an ensemble model. The essential steps of a boosting procedure can be summarised as follows [7]:

1. $\bar{f}_0(x) = 0$;
2. For $j = 1 : M$ do:
   (a) $(c_j, f_j(x)) = \arg\min_{c_j^*, f_j^*(x)} \sum_{i=1}^{N} L(y_i, \bar{f}_{j-1}(x_i) + c_j^* f_j^*(x_i))$
   (b) $\bar{f}_j(x) = \bar{f}_{j-1}(x) + c_j f_j(x)$
3. EndFor
4. $\bar{f}(x) = \bar{f}_M(x) = \sum_{j=1}^{M} c_j f_j(x)$.

Figure 1. A general boosting framework

Here $f_j^*(x)$ is the initial base learner derived from the data subset $\mathcal{D}^j$, $f_j(x)$ is optimized one by *boosting stage 2(a)* at the j-th iteration and c_j denotes the corresponding coefficient in the ensemble model $\bar{f}(x)$. The loss $L(y, \bar{f})$ can be any differential function which corresponds to different boosting algorithms. The most prominent example is *Adaboost* which just employ the exponential loss $L(y, \bar{f}) = \exp\{-y\bar{f}\}$. Other examples include *Logitboost* with the (minus) binomial log-likelihood $L(y, \bar{f}) =$

[2] Note that, in the context of boosting, we use $L(\cdot, \cdot)$ denote the loss function instead of $V(\cdot, \cdot)$ as done in kernel models.

$\log(1+\exp\{-y\bar{f}\})$ and L_2*boost* with the simple squared loss $L(y,\bar{f}) = \frac{1}{2}(y-\bar{f})^2$.

In contrast to the objective function (3) used in kernel models, we note that boosting framework in Figure 1 does not include a regularization term. Actually, boosting utilizes an empirical method, i.e., so-called "shrinkage" in statistics, to avoid the resulting overfit problem by a large number of base learners. This can be accomplished by replacing *stage 2(b)* in Figure 1 with

$$\bar{f}_j(x) = \bar{f}_{j-1}(x) + (\nu \cdot c_j) f_j(x), \tag{5}$$

where $0 < \nu \leq 1$ is a shrinkage factor. This can greatly improve the generalization performance of the algorithm [6]. The cost paid for better performance is *increased computation* since a large number of base learners are required. This conflicts with our expected objective of reducing computational cost. Another potential alternative to regularization in the context of boosting is introducing a penaly term in the objective, for example, the commonly used norm-2 penalty $\mu \sum c_j^2$, where $\mu > 0$ is a trade-off parameter. The explicit shortcoming of this strategy lies in introducing an additional parameter μ. No doubt, it will increase the computational burden of combining procedure for determining μ.

2.2 A new regularization for boosting

We introduce a different regularization term which is similar to the one employed by kernel models. The idea was inspired by the *sparse formulation* of kernel models (3), where the "*sparseness*" means that some entries of the solution α are exactly zeros. Let $\alpha_M = [\alpha_{i_1}, ..., \alpha_{i_M}]$ denote all the non-zero entries of α indexed by $I_M = \{i_1, ..., i_M\}$ which correponds to the set of so-called *basis vectors*, the sparse formulation of (3) is given by the optimization problem

$$\min_{\alpha_M} H(\alpha_M) = \frac{1}{N}\sum_{i=1}^{N} V(y_i, (K_M\alpha_M)_i) + \frac{\lambda}{2}\alpha_M^\top K_{MM}\alpha_M \tag{6}$$

and the resulting sparse model has a form of

$$f(x) = \sum_{j=1}^{M} \alpha_{i_j} k(\tilde{x}_j, x). \tag{7}$$

where $\{\tilde{x}_j \triangleq x_{i_j}, j = 1, ..., M\}$ are selected basis vectors, K_M is a $N \times M$ matrix of the kernel function $k(\cdot,\cdot)$ between all the training examples and selected basis vectors, i.e., $\{(K_M)_{ij} = k(x_i, \tilde{x}_j), i = 1, ..., N; j = 1, ..., M\}$ and K_{MM} is the $M \times M$ kernel matrix of evaluating basis vectors on the kernel function, i.e., $\{(K_{M,M})_{jj'} = k(\tilde{x}_j, \tilde{x}_{j'}), j, j' = 1, ..., M\}$.

Now we imagine $f(x)$ in (7) to be an ensemble model

$$\bar{f}(x) = \sum_{j=1}^{M} c_j f_j(x), \tag{8}$$

and now each term in the expansion is a base learner which consists of multiple kernel terms not just one as in (7), that is,

$$f_j(x) = \sum_{d=1}^{D} a_d^j k(\tilde{x}_d^j, x), \tag{9}$$

where D is the number of kernel terms [3] involved in the submodel $f_j(x)$, $\{\tilde{x}_d^j, d = 1, ..., D\} \subseteq \mathcal{D}^j$ are basis vectors and $\{a_d^j, d = 1, ..., D\}$ are weight parameters. Accordingly, the matrices K_M and K_{MM} in sparse formulation (3), denoted by F and Q respectively in the context of ensemble model, evolve as

$$F_{ij} = \sum_{d=1}^{D} a_d^j k(x_i, \tilde{x}_d^j), i = 1, ..., N; j = 1, ..., M \tag{10}$$

and

$$Q_{jj'} = \sum_{d=1}^{D}\sum_{d'=1}^{D} a_d^j a_{d'}^{j'} k(\tilde{x}_d^j, \tilde{x}_{d'}^{j'}), j, j' = 1, ..., M. \tag{11}$$

Finally, we can obtain the following objective function for combining multiple kernel models:

$$\min_c \bar{H}(c) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, (Fc)_i) + \frac{\mu}{2} c^\top Q c, \tag{12}$$

where $c = [c_1, \ldots, c_M]^\top$ is the vector of weighting M individual models built on different training subsets, μ is a regularization parameter, the components of matrices F and Q have been defined by (10) and (11), respectively. Since our ensemble objective function (12) keeps the same form as the objective (6) of one single kernel model, it is reasonable to set the parameter $\mu = \lambda$. By now, we have successfully introduced a new type of regularization term for constructing an anti-overfit ensemble model. We name the boosting technique with this new regularization method as "*kernel-boosting*".

In line with the Friedman's boosting framework, we propose our new general boosting framework for combining kernel models, which is described in Figure 2. Note that, in *stage 2(b)*, we have defined $\alpha^j = [\alpha_1^j, ..., \alpha_D^j]^\top$, $c^{j-1} = [c_1, ..., c_{j-1}]^\top$,

$$f_j = \begin{bmatrix} \sum_d \alpha_d^j k(\tilde{x}_d^j, x_1) \\ \sum_d \alpha_d^j k(\tilde{x}_d^j, x_2) \\ \vdots \\ \sum_d \alpha_d^j k(\tilde{x}_d^j, x_N) \end{bmatrix}_{N\times 1}, \tag{13}$$

[3] For brevity, we have assumed that all base learners have the same number of kernel terms or basis vectors.

1. $\bar{f}_0(x) = 0$ and $r^0 = y$;
2. For $j = 1 : M$ do:
   (a) Generate the j-th initial base learner $f_j^*(x) = \sum_{d=1}^{D} \alpha_d^j k(\tilde{x}_d^j, x)$ from the subset $\mathcal{D}^j$. Note that the targets of $\mathcal{D}^j$ are replaced with the current residual r^{j-1}.
   (b) $(c_j, a^j) = \arg\min_{c_j^*, \alpha^j} \left\{ \sum_{i=1}^{N} L(y_i, F_{j-1}c^{j-1} + c_j^* f_j) + \frac{\mu}{2} \begin{bmatrix} c^{j-1} \\ c_j^* \end{bmatrix}^\top \begin{bmatrix} Q_{j-1} & q_j \\ q_j^\top & q_j^* \end{bmatrix} \begin{bmatrix} c^{j-1} \\ c_j^* \end{bmatrix} \right\}$.
   (c) Compute the optimized base learner $f_j(x) = \sum_{d=1}^{D} a_d^j k(\tilde{x}_d^j, x)$.
   (d) $\bar{f}_j(x) = \bar{f}_{j-1}(x) + c_j f_j(x)$ and $r^j = r^{j-1} - c_j f_j$.
3. EndFor
4. $\bar{f}(x) = \bar{f}_M(x) = \sum_{j=1}^{M} c_j f_j(x)$.

Figure 2. A framework for boosting kernel models

$$q_j = \begin{bmatrix} \sum_d \sum_{d'} \alpha_d^1 \alpha_{d'}^j k(\tilde{x}_d^1, \tilde{x}_{d'}^j) \\ \sum_d \sum_{d'} \alpha_d^2 \alpha_{d'}^j k(\tilde{x}_d^2, \tilde{x}_{d'}^j) \\ \vdots \\ \sum_d \sum_{d'} \alpha_d^{j-1} \alpha_{d'}^j k(\tilde{x}_d^{j-1}, \tilde{x}_{d'}^j) \end{bmatrix}_{(j-1)\times 1} \quad (14)$$

and

$$q_j^* = \sum_d \sum_{d'} \alpha_d^j \alpha_{d'}^j k(\tilde{x}_d^j, \tilde{x}_{d'}^j), \quad (15)$$

and these three quantities f_j, q_j, q_j^* are all dependent on the parameters $\alpha_d^j, d = 1, ..., D$. The *step 2(a)* aims to generate an initial base learner from a random training subset and the current residual. The key *step 2(b)* can be efficiently solved by any gradient-based optimization algorithm.

In the next section, we will demonstrate how to apply this framework to combine multiple KRR models in the context of regression, but the *main* procedure is applicable to boost other kernel models.

3 Boosting Kernel Ridge Regression

In this section, we apply the framework of boosting kernel models described in the last section to KRR models for regression tasks. We firstly review some training algorithms to KRR on a single dataset. Secondly we discuss how to *efficiently* implement the *step 2(b)* in Figure 2 when boosting KRR models.

3.1 Solving single KRR

The problem of KRR is to minimize

$$\min_{\alpha} H(\alpha) = \frac{1}{2}\|y - K\alpha\|^2 + \frac{\lambda}{2}\alpha^\top K\alpha, \quad (16)$$

which is equivalent to solve the linear system of equations: $(K + \lambda I)\alpha = y$, which requires $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory. A feasible scheme to large data is to find a sparse estimate of α, i.e., solving

$$\min_{\alpha_D} H(\alpha_D) = \frac{1}{2}\|y - K_D\alpha_D\|^2 + \frac{\lambda}{2}\alpha_D^\top K_{DD}\alpha_D, \quad (17)$$

where we have changed the subscript M in (6) to D in order to highlight that M denotes the size of ensemble model and D is the size of each base learner. The optimal solution to sparse KRR (17) has a form of $\alpha_D = \Sigma_D K_D^\top y$ where $\Sigma_D = (K_D^\top K_D + \lambda K_{DD})^{-1}$. It is obvious that solving such a sparse solution only cost $\mathcal{O}(ND^2)$ time and $\mathcal{O}(ND)$ memory. But the trouble is how to pick up a good subset of basis vectors, which is a combinatorial problem in theory and has a crucial effect on generalization performance.

It has been shown that *forward selection algorithms* [17, 9, 19], which choose basis vector from all the remaining basis candidates one by one, could achieve good results in generalization while keeping the overall complexity to $\mathcal{O}(ND^2)$. The key step to forward approaches is the use of a criterion for deciding which training example should be chosen as basis vector at each iteration. A good criterion could achieve a good balance between computational cost and predictive performance. One path to achieve this is the criteria [9]

$$\Delta_i = \frac{1}{2}\frac{[k_i^\top (y - K_D\alpha_D) - \lambda \tilde{k}_i^\top \alpha_D]^2}{\lambda k_{i,i} + k_i^\top k_i}, \quad (18)$$

with

$$k_i = [k(x_1, x_i), ..., k(x_N, x_i)]^\top, k_{i,i} = k(x_i, x_i),$$
$$\tilde{k}_i = [k(\tilde{x}_1, x_i), ..., k(\tilde{x}_D, x_i)]^\top,$$

which measures the score of the training instance x_i to be selected in the current iteration. This evaluating method requires $\mathcal{O}(N)$ for evaluating one instance. At each iteration, if we compute this score for only κ candidates from all the remaining instances, then the complexity for basis selections is $\mathcal{O}(\kappa ND)$ till D basis vectors are included. Typically, the value of κ is set to 59 [17] or the predefined number D of selected basis vectors [9]. The linear scaling in N makes this selection scheme viable for large problems.

Another criterion is *info-gain* method [16] which is always inferior to the criteria (18) for generalization on large datasets [9]. But this method evaluates the score of adding one case just in $\mathcal{O}(1)$ time, which makes this algorithm almost as fast as using random selection. To reduce the computational cost to a great extent, our boosting procedure use this fast approach to initialize the base learner $f_j^*(x)$ (see *step 2(a)* in Figure 2).

In the experimental section, the algorithm with the criteria (18) running on the whole training data will be compared empirically to our ensemble method both in computational time and generalization performance.

3.2 Details of boosting KRR

We are going to boost a set of regression models, so the loss function $L(y, \bar{f})$ in our ensemble objective (12) will be replaced with the squared function. Correspondingly, the step *2(b)* can be further detailed as

$$\bar{H}(c_j^*,\alpha^j)=\min_{c_j^*,\alpha^j}\left\{\frac{1}{2}\left\|r^{j-1}-c_j^* f_j\right\|^2+ \frac{\mu}{2}\begin{bmatrix}c^{j-1}\\ c_j^*\end{bmatrix}^\top\begin{bmatrix}Q_{j-1}\, q_j\\ q_j^\top\;\; q_j^*\end{bmatrix}\begin{bmatrix}c^{j-1}\\ c_j^*\end{bmatrix}\right\}, \tag{19}$$

where $r^{j-1} = y - F_{j-1}c^{j-1}$ is the residual. Since the condition for optimality of c_j^* is

$$\frac{\partial \bar{H}(c_j^*, \alpha^j)}{\partial c_j^*} = c_j^*(\mu q_j^* + f_j^\top f_j) + [\mu q_j^\top c^{j-1} - f_j^\top r^{j-1}] = 0,$$

we can get

$$c_j^* = \frac{f_j^\top r^{j-1} - \mu q_j^\top c^{j-1}}{\mu q_j^* + f_j^\top f_j}. \tag{20}$$

After simplification of some notations, we can show that the problem (19) is equivalent to

$$\bar{H}(\alpha^j)=\bar{H}_{j-1}-\left\{\frac{1}{2}\frac{[f_j^\top r^{j-1}-\mu q_j^\top c^{j-1}]^2}{\mu q_j^*+f_j^\top f_j}\right\}. \tag{21}$$

The derivative of (21) w.r.t. $\alpha_d^j, d = 1, ..., D$, can be easily obtained, that is

$$\frac{\partial \bar{H}(\partial \alpha^j)}{\alpha_d^j} = \frac{1}{2}c_j^*[2(\dot{f}_j^\top r^{j-1} - \mu \dot{q}_j^\top c^{j-1}) - c_j^*(\mu \dot{q}_j^* + 2 f_j^\top \dot{f}_j)] \tag{22}$$

where

$$\dot{f}_j = \frac{\partial f_j}{\partial \alpha_d^j}, \dot{q}_j = \frac{\partial q_j}{\partial \alpha_d^j}, \dot{q}_j^* = \frac{\partial q_j^*}{\partial \alpha_d^j}.$$

It can be noted that the optimization problem (21) involves a quadratic function w.r.t. α^j and thus the global optimum can be reached in at most D iterations by conjugate gradient (CG) method.

Another issue *step 2(d)* is how to compute the optimal weight c^j from old weight c^{j-1} and new entry c_j. In our general framework for boosting kernel models, we simply use $c^j = [c^{j-1\top}, c_j]^\top$ with no changes for c^{j-1}. In fact, another better way is to update the weights of all the base learners included so far. In the context of boosting KRR models, this can be done very efficiently in $\mathcal{O}(NM^2)$ by a recursive formula (see Appendix for details).

The final version of boosting KRR models can now be summarized as follows:

1. Initialization:
   (a) size of training subset (e.g., 2000);
   (b) number D of selected basis vectors for each base learner (e.g., $D = 200$);
   (c) maximal number M of base learners to be boosted (e.g., $M = 150$);
   (d) iteration number of CG method (e.g., 50);
2. $\bar{f}_0(x) = 0$ and $r^0 = y$;
3. For $j = 1 : M$ do
   (a) Generate a random subset $\mathcal{D}^j$ from training data and r^{j-1}, then produce the j-th base learner by employing *info-gain* method;
   (b) Obtain the optimal solution a^j to (21) by the CG method;
   (c) Update c^{j-1}, r^{j-1} to c^j, r^j, respectively, based on (27);
   (d) Compute the optimized base learner $f_j(x) = \sum_{d=1}^{D} a_d^j k(\tilde{x}_d^j, x)$.
4. EndFor
5. Output the ensemble model $\bar{f}(x) = \sum_{j=1}^{M} c_j^M f_j(x)$.

The major computational cost incurred in our boosting approach is the step of optimizing the base learner (e.g. *the stage 3(b)*), at each iteration. Note that evaluating the gradient information $\dot{f}_j, \dot{q}_j, \dot{q}_j^*$ need $\mathcal{O}(ND)$ time. So the overall complexity would be $\mathcal{O}(NDM)$ time and $\mathcal{O}(ND)$ memory. We set the size D of each base learner to 200 and the size M of ensemble model to 150 in this paper. Unlike single KRR model on the whole dataset, it costs $\mathcal{O}(ND^2)$ time and $\mathcal{O}(ND)$ memory *but where D is significantly larger than* 200 especially for large-scale datasets.

4 Numerical Experiments

In this section, we empirically demonstrate that the proposed boosting appoach for combining multiple KRR models could achieve better generalization performance and lower computational requirements simultaneously when compared to training a single KRR with (18). The following data sets are used in the next evaluations:

1. Outaouais: The data was used in the "Evaluating Predictive Uncertainty Challenge " but its background information is not available [4].

2. Kin40k: This dataset represents the forward dynamics of an 8 link all-revolute robot arm, the task is to predict the distance of the end-effector from a target, given the twist angles of the 8 links as features [5].

3. Sarcos: The task is related to learn the inverse dynamics of a seven degrees-of-freedom SARCOS anthropomorphic robot arm [6].

4. Census-house: The task is to predict the median price of the house based on some certain demographic information [7].

Some properties of these data sets are shown in Table 1.

Table 1. Properties of four datasets

Data set	#training	#testing	#dimension
Outaouais	20,000	9,000	37
Kin40k	36,000	4,000	8
Sarcos	44,484	4,449	21
Census-house	20,000	2,784	138

To evaluate prediction performance, we utilize *normalized mean square error* (NMSE) given by $\frac{1}{N_{\text{test}}}\sum_i \frac{y_i - f(x_i)}{\text{Var}(y)}$, where N_{test} is the number of test examples and $\text{Var}(y)$ is the standard deviation of training targets. The algorithms presented in this section are coded in Matlab 7.0 and all the numerical experiments are conducted on a 2G Pentium-4 machine with 512M RAM, running Windows XP.

For the first three datasets, the *squared-exponential* kernel function was used [9],

$$k(x_i, x_j; \theta) = \theta_0 \exp\left(-\frac{1}{2}\sum_{l=1}^{L}\theta_l(x_{i,l} - x_{j,l})^2\right) + \theta_b, \tag{23}$$

where $\theta_0, \theta_l, \theta_b > 0$ are hyperparameters, L is the dimension and $x_{i,l}$ is the l-th entry of the input x_i. Since Gaussian process regression (GPR) model can be regarded as a Bayesian interpretation of KRR [12], we can maximize the marginal likelihood of GPR on a random training subset (e.g., 1000 examples) for optimal settings of the hyperparameters $\theta_0, \theta_l, \theta_b$ and regularization parameter λ. This task can be done by routines of the well-known NETLAB software [8]. Since Census-house is a high dimensional data, optimizing much more hyperparameters will take a very long time. So we simply use the Gaussian kernel $\exp(-\|x - x'\|^2/\theta)$, with $\theta = \frac{1}{N}\sum_{i,j=1}^{N}\|x_i - x_j\|^2$. The associated λ parameter is tuned on a small validation dataset.

For the proposed boosting ensemble approach, we fix the size of training subset on 2000, the size of each learner $D = 200$ and the upper limit of ensemble model size $M = 150$ for all datasets considered. When implementing the forward selection algorithm with the criteria (18) on the whole data, we vary the maximal allowed number of selected basis vectors for different datasets due to the limit of memory size. Generally, the more basis vectors chosen the better generalization performance we could achieve. The parameter κ involved in the selection process is set to 59 in order to further save the storage space for training a single KRR model.

To remove the randomness involved in the algorithms, the results reported below are averaged on 10 independant runs. Figure 3-6 illustrate the results of test NMSE as a function of CPU time by learning a single model and boosted ensembles on four large-scale datasets, respectively. It is clear that training multiple smaller KRR models by our proposed approach consistently achieve much better generalization performance than training a single KRR model if the same CPU time is gone. Moreover, we can note that the curves for single KRR models terminate quite earlier than ensemble models. This is because that the programs cannot proceed due to an out-of-memory error at this point. Table 2 shows the maximal allowed number of selected basis vectors for single KRR on four datasets. If considering that the setting of this number for each base learner in boosted ensembles is just 200, we can understand why our ensembles has the advantage of requiring less memory than single large model.

We also summarize the best results of test NMSE obtained by single model and boosted ensembles under the parameter settings described above in Table 3. For all the datasets considered, ensembles is significantly better than single KRR models on generalization performance, which is dicided by a p-value threshold of 0.001 in the paired t-test. Table 4 reports comparisons of CPU time elapsed when

[4] `http://predict.kyb.tuebingen.mpg.de/datasets/`
[5] `http://ida.first.fraunhofer.de/~anton/data/`
[6] `http://www.gaussianprocess.org/gpml/data/`
[7] `http://www.cs.ust.hk/~jamesk/data/census.zip`

[8] It is available at `http://www.ncrg.aston.ac.uk/netlab/index.php`.

Table 2. The maximal allowed number of selected basis vectors for single KRR on four datasets

Data set	# selected basis
Outaouais	1,160
Kin40k	680
Sarcos	560
Census-house	1,040

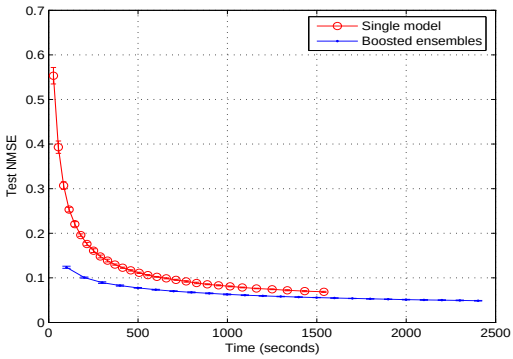

Figure 3. NMSE comparisons of single model and ensembles for the Outaouais dataset as a function of CPU time.

single model and ensembles reach the same predictive accuracy. It can be seen that boosted ensembles is at least two times faster than training single KRR model. The number in parentheses reflects the number of added base learners in ensembles.

Table 3. Comparisons of best NMSE obtained by single model and ensembles.

Data set	single KRR	boosted ensembles
Outaouais	0.0686±0.0013	0.0463±0.0005
Kin40k	0.0280±0.0006	0.0176±0.0002
Sarcos	0.0181±0.0001	0.0155±0.0001
Census-house	0.0751±0.0003	0.0738±0.0001

5 Conclusions

We presented a general boosting framework to combine multiple kernel models where each one is initialized from an individual training subset. The proposed framework

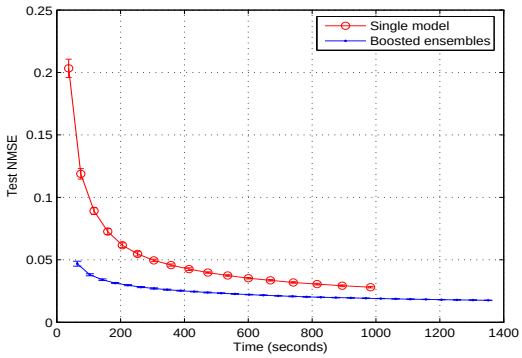

Figure 4. NMSE comparisons of single model and ensembles for the Kin40k dataset as a function of CPU time.

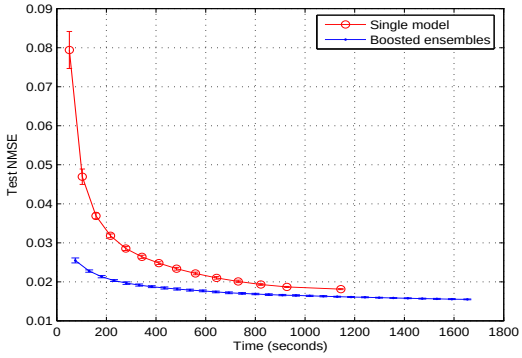

Figure 5. NMSE comparisons of single model and ensembles for the Sarcos dataset as a function of CPU time.

was employed to boost multiple KRR models for regression tasks. The resulting ensembles is empirically compared to the state-of-the-art algorithm for learning a single KRR model on three large-scale regression problems. The results support that we still benefit a lot from ensembling multiple kernel models at least for regression even though the base learners are stable.

In the near future, we will investigate the performance of applying the proposed boosting framework to large-scale classification problems. Another interesting direction is to extend our work to Multiple Kernel Learning (MKL) [18] which aims to address the issue of model selection and heterogeneous property involved.

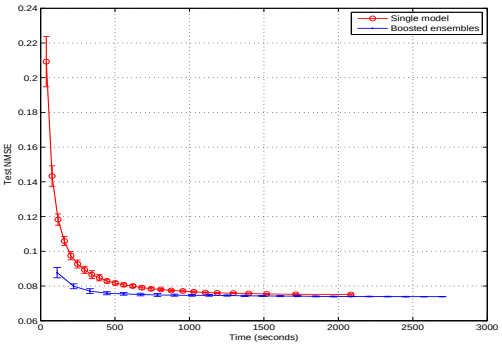

Figure 6. NMSE comparisons of single model and ensembles for the Census-house dataset as a function of CPU time.

Table 4. Comparisons of CPU time when single model and ensembles reach the same accuracy.

Data set	single KRR	boosted ensembles
Outaouais	1540	820 (40)
Kin40k	983	264 (30)
Sarcos	1144	485 (45)
Census-house	2083	672 (30)

Appendix

1. The gradients of f_j, q_j, q_j^* w.r.t. $\alpha_d^j, d = 1, ..., D$ can be detailed as

$$\dot{f}_j = \frac{\partial f_j}{\partial \alpha_d^j} = \begin{bmatrix} k(\tilde{x}_d^j, x_1) \\ k(\tilde{x}_d^j, x_2) \\ \vdots \\ k(\tilde{x}_d^j, x_N) \end{bmatrix} \tag{24}$$

$$\dot{q}_j = \frac{\partial q_j}{\partial \alpha_d^j} = \begin{bmatrix} \sum_{d'} \alpha_{d'}^1 k(\tilde{x}_{d'}^1, \tilde{x}_d^j) \\ \sum_{d'} \alpha_{d'}^2 k(\tilde{x}_{d'}^2, \tilde{x}_d^j) \\ \vdots \\ \sum_{d'} \alpha_{d'}^{j-1} k(\tilde{x}_{d'}^{j-1}, \tilde{x}_d^j) \end{bmatrix} \tag{25}$$

$$\dot{q}_j^* = \frac{\partial q_j^*}{\partial \alpha_d^j} = 2 \sum_{d'} \alpha_{d'}^j k(x_{d'}^j, x_d^j), \tag{26}$$

which can be evaluated in $\mathcal{O}(ND)$ complexity.

2. In order to update c^{j-1}, r^{j-1} to c^j, r^j, we define the following notations. Let L_j be the factor of Cholesky factorisation: $L_j L_j^\top = Q_j$, let $G_j = F_j L_j^{-\top}$ and M_j be the factor of another Cholesky decomposition: $M_j M_j^\top = (G_j^\top G_j + \lambda I d_j)$, we have

$$c^j = L_j^{-\top}(M_j M_j^\top)^{-1} G_j^\top y \quad \text{and} \quad r^j = y - F_j c^j.$$

The involved recursive steps can be summarized:

$$l_j = L_{j-1}^{-1} q_j, l_j^* = \sqrt{q_j^* - l_j^\top l_j}, g_j = \frac{f_j - G_{j-1} l_j}{l_j^*},$$

$$m_j = M_{j-1}^{-1}(G_{j-1}^\top g_j), \eta = M_{j-1}^{-\top} m_j, d_j = g_j - G_{j-1}\eta,$$

$$b = d_j^\top y, c = d_j^\top g_j, m_j^* = \sqrt{\lambda + c}, c_j = \frac{b}{l_j^*(\lambda + c)},$$

$$c^j = \begin{bmatrix} c^{j-1} - c_j[L_{j-1}^{-\top}(l_j + l_j^*\eta)] \\ c_j \end{bmatrix}, r^j = r^{j-1} - \frac{b d_j}{\lambda + c}, \tag{27}$$

and finally

$$L_j = \begin{bmatrix} L_{j-1} & 0 \\ l_j^\top & l_j^* \end{bmatrix}, M_j = \begin{bmatrix} M_{j-1} & 0 \\ m_j^\top & m_j^* \end{bmatrix}, G_j = [G_{j-1} \;\; g_j].$$

Since the matrices L_j and M_j are low-triangular, the multiplication of their inverse and a vector can be computed very efficiently.

References

[1] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[2] V. V. C. Saunders, A. Gammermann. Ridge regression learning algorithm in dual variables. In J. Shavlik, editor, *ICML-1998*, San Francisco, CA, 1998. Morgan Kaufmann.

[3] A. Choudhury, P. B. Nair, and A. J. Keane. A data parallel approach for large-scale gaussian process modeling. In *Proceedings of SDM-2002*, 2002.

[4] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of svms for very large scale problems. *Neural Computation*, 14(5), 2002.

[5] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. In *Advances in Large Margin Classifiers*, 2000.

[6] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[7] J. H. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–407, 2000.

[8] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade svm. In *NIPS-17*. MIT Press, 2005.

[9] S. S. Keerthi and W. Chu. A matching pursuit approach to sparse Gaussian process regression. In *NIPS-18*. MIT Press, 2006.

[10] S. Mika. *Kernel Fisher Discriminants*. PhD thesis, University of Technology, Berlin, Germany, October 2002.

[11] D. Pavlov, J. Mao, and B. Dom. Scaling-up support vector machines using boosting algorithm. In *Proceedings of ICPR-2000*, 2000.

[12] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[13] B. Scholkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *Proceedings of the 14th Annual Conference on Computational Learning Theory*, pages 416–426, 2001.

[14] B. Scholkopf, A. Smola, and K.-R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[15] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2002.

[16] M. Seeger, C. K. I. Williams, and N. D. Lawrence. Fast forward selection to speed up sparse gaussian process regression. In *Ninth International Workshop on Artificial Intelligence and Statistics*, Key West, Florida, 2003.

[17] A. J. Smola and B. Schökopf. Sparse greedy matrix approximation for machine learning. In *ICML-2000*, pages 911–918, 2000.

[18] S. Sonnenburg, G. Ratsch, and C. Schafer. A general and efficient multiple kernel learning algorithm. In *NIPS-18*. MIT Press, 2006.

[19] P. Sun and X. Yao. Greedy forward selection algorithms to sparse Gaussian Process Regression. In *Proceedings of IJCNN-2006*, 2006. to appear.

[20] V. Tresp. A bayesian committee machine. *Neural Computation*, 12:2719–2741, 2000.

[21] V. Tresp. Scaling kernel-based systems to large data sets. *Data Mining and Knowledge Discovery*, 5(3):197–211, 2001.

[22] M. Yamana, H. Nakahara, M. Pontil, and S. Amari. On different ensembles of kernel machines. In *Proceedings of ESANN-2003*, 2003.

[23] J. Zhu and T. Hastie. Kernel logistic regression and the import vector machine. *Journal of Computational and Graphical Statistics*, 14(1):185–205, 2005.