# MARK: A Boosting Algorithm for Heterogeneous Kernel Models

Kristin P. Bennett
Department of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY 12180
bennek@rpi.edu

Michinari Momma
Department of Decision Sciences and Engineering Systems
Rensselaer Polytechnic Institute
Troy, NY 12180
mommam@rpi.edu

Mark J. Embrechts
Department of Decision Sciences and Engineering Systems
Rensselaer Polytechnic Institute
Troy, NY 12180
embrem@rpi.edu

## ABSTRACT

Support Vector Machines and other kernel methods have proven to be very effective for nonlinear inference. Practical issues are how to select the type of kernel including any parameters and how to deal with the computational issues caused by the fact that the kernel matrix grows quadratically with the data. Inspired by ensemble and boosting methods like MART, we propose the Multiple Additive Regression Kernels (MARK) algorithm to address these issues. MARK considers a large (potentially infinite) library of kernel matrices formed by different kernel functions and parameters. Using gradient boosting/column generation, MARK constructs columns of the heterogeneous kernel matrix (the base hypotheses) on the fly and then adds them into the kernel ensemble. Regularization methods such as used in SVM, kernel ridge regression, and MART, are used to prevent overfitting. We investigate how MARK is applied to heterogeneous kernel ridge regression. The resulting algorithm is simple to implement and efficient. Kernel parameter selection is handled within MARK. Sampling and "weak" kernels are used to further enhance the computational efficiency of the resulting additive algorithm. The user can incorporate and potentially extract domain knowledge by restricting the kernel library to interpretable kernels. MARK compares very favorably with SVM and kernel ridge regression on several benchmark datasets.

## 1. INTRODUCTION

Support Vector Machines (SVMs) and other Kernel Methods have proven to be very effective inference tools for many applications [20]. By introducing kernels, many linear methods for classification, regression and unsupervised learning can be transformed into nonlinear methods [18]. For each application, an appropriate kernel and any associated parameters must be selected.

The choice of kernel represents an opportunity to include background knowledge into the modeling process and to bias the models to have desirable properties. The kernel determines how similarity is measured on a particular domain. For any given application, there are many possible kernels. Typical algorithms require that the user pick one type of kernel. Then the kernel parameters have to be chosen so as to obtain a model with good generalization capability. Kernel parameter selection typically optimizes cross validation accuracy or leave-one-out bounds [2, 14]. However, these method can suffer from the "curse of dimensionality" severely when optimizing large number of kernel parameters. Thus one sticks to kernels such as the Radial Basis Function (RBF) kernels that can approximate many functions and have only one parameter. But then the only opportunity for incorporating domain knowledge is through the choice of input attributes. The resulting RBF models may provide very good predictions, but they provide little information about the underlying properties of the problem, e.g. which attributes are important etc.

Our goal is to develop an approach to construct inference models based on heterogeneous families of kernels. We focus on the regression problem, but the general approach is applicable to classification and other types of inference problems. Based on domain knowledge and desired properties of the final regression function, the application specialist can choose a library of kernels. The regression algorithm should resolve which kernels (including their parameters) to include and efficiently construct the heterogeneous kernel function. The resulting function should generalize well and not require storage of large amounts of data in order to make future predictions.

The proposed multiple additive regression kernel approach, MARK, accomplishes these goals. Gradient descent approaches developed for ensemble methods such as AdaBoost, MarginBoost, gradient boost, leveraging [6, 12, 7, 9, 5, 15] are generalized to the heterogeneous kernel model problem. The columns of the heterogeneous kernel matrix are generated on the fly, just like base learners are used to construct hypotheses in ensembles. Regularization is used to prevent overfitting and to enforce sparsity within the kernel representation. Analogously to how ensemble methods can beneficially build upon "weak learners", MARK can be applied to families of "weak kernels" that would not be adequate for an algorithm using a single kernel. Computational results demonstrate that MARK efficiently produces sparse heterogeneous kernel models based on simple interpretable weak kernels that generalize as well, and require dramatically less storage and testing time than baseline kernel methods produced using a single kernel.

This paper is organized as follows. In Section 2, we examine the heterogeneous kernel model. A regularized loss function is minimized to construct a model. We examine the gradient of this loss function and discuss coordinate descent strategies for optimizing the model. In Section, 3 we demonstrate how MARKing can be applied to least squares kernel models for regression which are known by various names including least squares SVM [19] and Kernel Ridge Regression (KRR) [17]. We review the basic KRR method. Then propose two different approaches for heterogeneous KRR, MARK-L and MARK-S. In Section 4, we introduce the idea of a "weak kernel" functions. In Section 5, the computational costs of MARK and strategies for improving it are examined. Computational results are also provided in Section 5. We conclude with a discussion of the approach and future research issues.

## 2. HETEROGENEOUS KERNEL MODELS

We define the heterogeneous kernel model (HKM) problem as follows. Giving a training data set consisting of $N$ $M$-dimensional points $x_i$ with associated output $y_i$, we would like to construct a regression function $F$ that minimizes some convex loss function $L(y, F)$ on the training data. To prevent overfitting the function $F$ is regularized by the convex function $P(F)$. The regularization function $P(F)$ helps to constrain the possible set of regression functions $F$ considered. Typically $P(F)$ is chosen such that some norm of the parameters of $F$ is minimized. Thus our goal is to find the solution of

$$min_F \quad \sum_{i=1}^N L(y_i, F(x_i)) + CP(F) \tag{1}$$

where $C \geq 0$ is a fixed parameter. With the choice of appropriate loss functions, these general models are applied to both classification and regression problems, but we focus on regression in this paper.

For a single kernel problem like SVM, $F$ is based on a single kernel $K$ and would be defined as

$$F(x) =: f_{\alpha,b} = \sum_{i=1}^{N} \alpha_i K(x, x_i) + b. \tag{2}$$

For example, $K$ could be a radial basis function kernel with a fixed parameter $\sigma$:

$$K(x, x_i) = e^{-\frac{||x-x_i||^2}{\sigma}}$$

where $|| \cdot ||$ is the Euclidean or 2-norm. For HKM, the function $F$ is composed of a linear combination of heterogeneous kernel functions $K_1, \dots, K_q$

$$F(x) =: f_{\alpha,b} = \sum_{q=1}^{Q} \sum_{i=1}^{N} \alpha_{qi} K_q(x, x_i) + b. \tag{3}$$

The kernels can be of any type and they need not all be of the same type. For example, each $K_q$ could be a RBF kernel with a different parameter $\sigma_q$.

If $P(F) = p(\alpha, b)$ for some convex function $p$, then the HKM problem becomes

$$\begin{array}{ll} min_{\alpha,b} & H(\alpha, b) := \sum_{i=1}^N L(y_i, f_{\alpha,b}(x_i)) + Cp(\alpha, b) \\ with & f_{\alpha,b} = \sum_{q=1}^Q \sum_{i=1}^N \alpha_{qi} K_q(x, x_i) + b. \end{array} \tag{4}$$

Note that the size of $\alpha$ is very large $(Q \times N)$ since there is a column of each kernel corresponding to each data point. So it is not immediately evident that it is practical to solve this problem. Certainly we cannot afford to generate the full heterogeneous kernel matrix, $K = [K_1, \dots, K_Q]$.

But in fact, those familiar with boosting and ensemble methods will realize that the problem is just like the optimization problem solved in ensemble approaches [5, 9, 10, 12]. Ensemble methods generate linear combination of hypotheses. We can think of each column of the kernel as a hypothesis. Just like ensemble methods generate hypothesis on the fly, we can generate kernel columns on the fly. The generic gradient-based ensemble algorithm such as gradient boosting used in MART, Margin Boost, and Leveraging can be readily adapted to the heterogeneous kernel problem. The key to these approaches is the gradient of the loss function.

Most ensemble algorithms can be generated using gradient descent methods in function space or coordinate-descent in the space of $\alpha, b$. Assuming the loss and regularization functions are continuously differentiable, the gradient of $H$ with respect to $f$ is

$$\frac{\partial H}{\partial f} = \sum_{i=1}^{N} \frac{\partial L\left(y, f(x_i)\right)}{\partial f(x_i)} \tag{5}$$

By the chain rule the gradient of $H$ with respect to $\alpha_i$ is

$$\begin{array}{rcl} \frac{\partial H}{\partial \alpha_{qj}} & = & \sum_{i=1}^N \frac{\partial L(y, f(x_i))}{\partial f(x_i)} \frac{\partial f(x_i)}{\partial \alpha_{qj}} + C \frac{\partial p(\alpha,b)}{\partial \alpha_{qj}} \\ & = & \sum_{i=1}^N \frac{\partial L(y, f(x_i))}{\partial f(x_i)} K_q(x_i, x_j) + C \frac{\partial p(\alpha,b)}{\partial \alpha_{qj}} \end{array} \tag{6}$$

Coordinate descent algorithms optimize one component of the solution vector at a time. If $\nabla H(\alpha, b) = 0$ then the current solution is optimal. If $\frac{\partial H(\alpha,b)}{\partial \alpha_{qj}} \neq 0$ then either decreasing or increasing $\alpha_{qj}$ (depends on the sign of the partial gradient) will strictly decrease $H$. If $p = \frac{1}{2}\|\alpha\|^2$ and all the $\alpha_{qj} = 0$, then the best kernel column to choose is the one maximizing

$$\left| \sum_{i=1}^N \frac{\partial L(y, f(x_i))}{\partial f(x_i)} K_q(x_i, x_j) \right| \tag{7}$$

This is roughly the column that most highly correlates with the gradient.

MARKing works by computing the gradient for the current kernel ensemble $F_{\alpha^t,b}$, generating the column that maximizes (7), adding the column to the kernel submatrix $K^t$ and then computing the optimal weights $\alpha_t, b$ for the heterogeneous kernel, then repeating until the algorithm converges or the maximum iteration limit is reached.

There are two choices on how to compute the optimal weights $\alpha_t, b$. The usual ensemble and additive model approach is to just update the component of $\alpha_t$ corresponding to the newly marked kernel column. The second approach is to update the weights of all the kernels selected and marked so far. Both approaches have pros and cons that are somewhat dependent on the choice of loss function. Thus for the remainder of the paper we will focus on MARKing for a particular loss function, squared error in regression problem.

## 3. MARKING KRR

In this section, we assume a least square error loss function for regression with 2-norm regularization. Without kernels, such a problem is usually called ridge regression [9]. With kernels, we will call the problem kernel ridge regression although it is sometime referred to by other names [4]. For a single kernel function $K_1$, we can define the kernel matrix $(K_1)_{ij} = K_1(x_i, x_j)$, so for $N$ training points, $K_1$ is a $N \times N$ matrix. The kernel ridge regression problem is then

$$\min H(\alpha, b) = \frac{C}{2}\left(\alpha'\alpha + b^2\right) + \frac{1}{2}\|y - K\alpha - be\|_2^2 \qquad (8)$$

where $\alpha \in R^N$, $b \in R$, $C \geq 0$ is a constant and $e$ is a $N$ dimensional vector of ones.

Kernel regression is very attractive because optimization problem (8) can be solved by simply solving a system of equalities. Since the problem is an unconstrained convex optimization problem, the necessary and sufficient condition for optimality is $\nabla H(\alpha, b) = 0$. Thus by doing an analysis such as in [8], the solutions satisfies

$$\left[G'G + CI\right]\begin{bmatrix}\alpha \\ b\end{bmatrix} = G'y, \qquad (9)$$

where $G = [K\ e]$.

If $(\hat{\alpha}, \hat{b})$ solve equation (9), then the final regression model is

$$f(x) = \sum_{i=1}^{N} \hat{\alpha}_i K(x, x_i) + \hat{b} \qquad (10)$$

Unlike classic SVM or least squares model with LASSO type regularization, this optimization solution can be found by simply solving a system of equations, without introducing an optimization package. The catch is that the resulting optimal $\hat{\alpha}$ is not sparse, e.g. $\hat{\alpha}_i \neq 0$ for many $i$. By choosing alternative objective functions such as in SVM, only relatively few multipliers for the kernel columns, the support vectors, will be nonzero. KRR has the advantage of simple implementations and relatively fast training times. The method can be applied to any type of kernel function. There are no limitations on the kernel matrix $K$. In SVM, the kernel matrix is assumed to be a square positive definite matrix. Typically the choice of kernels is limited to those that obey Mercer's Condition since this ensures that $K$ is positive definite (see [20]). In the approach studied here, the matrix $K$ need not be positive definite or square. Kernel functions that do not satisfy Mercer's condition may be utilized. But the disadvantage of KRR without MARKING is that all the training data has to be stored in order to test new data.

### 3.1 Heterogeneous KRR

The flexibility in the choice of $K$ in KRR allows the model to be easily extended to Heterogeneous kernel ridge regression (HKRR), but the resulting optimization problem is more challenging. Define the heterogeneous kernel matrix $K$ as the concatenation of $Q$ kernel matrices $K_q \in R^{N \times N}$, $K = [K_1, \ldots, K_Q] \in R^{N \times QN}$. The above definition of the kernel ridge regression model (8) then applies to HKMs but now $\alpha \in R^{QN}$.

If we treat each column of $K$ as a hypothesis in an ensemble method, we can use ensemble algorithms to generate a small set of kernel columns/hypotheses that approximately solve the model. If we denote the $i^{th}$ column of $K$ as $K_i$ then

$$\frac{\partial H(\alpha,b)}{\partial \alpha_i} \quad = \quad -K_i'(y - (K\alpha + be)) + C\alpha_i \qquad (11)$$

The residual error vector for the data for any particular choice of $\alpha, b$ is $r = y - (K\alpha + be)$ , where $e$ is an $N$ vector of ones. By the above coordinate descent analysis, any column of $K$ with the maximum or almost maximum value of

$$|K_i' r - C\alpha_i| \qquad (12)$$

is a good candidate for inclusion in the model.

Motivated by the leveraging and column generation algorithms for classification in [15, 16], we developed the MARK-L algorithm described below. Note that we assume $\alpha_i^t = 0$ for any kernel column $K_i$ that has not been generated. MARK-L works by generating a kernel column to include in the model, concatenating that column to the kernel submatrix $\hat{K}$, and then solving the system of equalities (9) to find the optimal $\alpha^t, \hat{b}$ for the KRR model restricted to $\hat{K}$. At iteration $t$, MARK-L requires $(t+1) \times (t+1)$ storage and the solution of a system of equalities at each iteration. Since typically the maximum $T$ is small and we use fast iterative Krylov-type algorithms for solving the system of equalities, this is not an issue except on massive datasets. The primary benefit of solving the restricted KRR problem at each iteration is much faster convergence and therefore fewer kernel columns are needed to achieve good solutions. A secondary benefit of MARK-L is that it forces the regularization term to be applied at each iteration in order to prevent overfitting.

Algorithm 3.1. *MARK-L* $(X, y, T, tol)$

```
1. Select $b = \hat{y}$
2. Let $\hat{K} = \{\}$
3. Let $\alpha^0 = 0$
4. for $t := 0$ to $T$ do
5.     Let $r^t := y - \hat{K}\alpha^t - \hat{b}e$
6.     Generate kernel column $K_{t+1}$ for $X$ approximately
       maximizing $|K_{t+1}' r|$
7.     if $|K_{t+1}' r| \leq tol$ then
8.         return $\alpha^t, \hat{b}$
9.     end if
10.    $\hat{K} = [\hat{K}, K_{t+1}]$
11.    Solve (9) for $\alpha_t, \hat{b}$
12. end for
13. return $\alpha^t, \hat{b}$
```

Another approach is to use the more typical boosting or stepwise additive regression methods that for each iteration optimize the multiplier $\alpha_t$ for the incoming column and then fixes it for the remainder of the algorithm [6, 9]. Storage is drastically reduced because the restricted kernel submatrices need not be stored. Each iteration is less expensive because there is no need to solve a system of equalities. One disadvantage of this approach is regularization. The MART algorithm in this case applies a parameter called "shrinkage" to address this problem. Regularization or "flatness" is imposed by reducing the weight of each added kernel column by a constant factor $\nu$ at each iteration. As we will see, this can greatly improve the generalization performance of the algorithm. Thus our low storage version of MARK, MARK-S incorporates shrinkage $\nu$. However, MARK-S needs more kernel columns, or iterations, to converge because typically the shrinkage parameter is set to be small. Therefore, sparsity is severely lost in MARK-S.

ALGORITHM 3.2. *MARK-S* $(X, y, T, \nu, tol)$

```
1. Select $b = \hat{y}$
2. Let $\hat{K} = \{\}$
3. Let $\alpha_0 = 0$
4. Let $r^0 := y - \hat{b}e$
5. for $t := 0$ to $T$ do
6.     Generate kernel column $K_{t+1}$ for $X$ approximately maximizing $|K'_{t+1}r|$
7.     if $|K'_t r| \leq tol$ then
8.         return $\alpha, \hat{b}$
9.     end if
10.    Compute $\alpha_t = argmin \sum_{i=1}^{N} ||r^t - \alpha K_{t+1}||^2$
11.    Let $r^{t+1} := r^t - \nu\alpha_t K_{t+1}$
12. end for
13. return $\nu\alpha, \hat{b}$
```

## 3.2 Solving KRR Subproblem

The HKRR subproblem in step 11 of MARK-L can be solved quite efficiently. In fact we don't really store the matrix $\hat{K}$. Note that the only matrix required is $G'G$ (9). The storage required for $G'G$ is at most $(T+1) \times (T+1)$. To show this more explicitly, we express $G'G$ in terms of $\hat{K}$ and $e$ (a vector of ones);

$$G'G = \begin{bmatrix} \hat{K}'\hat{K} & \hat{K}'e \\ e'\hat{K} & e'e \end{bmatrix}. \tag{13}$$

When a kernel column $K_t$ is added at iteration $t$, the new submatrix $\hat{K}^{(t+1)'}\hat{K}^{(t+1)}$ will be written as;

$$\hat{K}^{(t+1)'}\hat{K}^{(t+1)} = \begin{bmatrix} \hat{K}'\hat{K} & \hat{K}'K_t \\ K'_t\hat{K} & K'_tK_t \end{bmatrix}. \tag{14}$$

Therefore, only update of $t+1$ elements from $\left(\hat{K}'K_t, K'_tK_t\right)$ is needed taking account of the symmetry. As for

$$\hat{K}^{(t+1)'}e = \begin{bmatrix} \hat{K}' \\ K'_t \end{bmatrix} e, \tag{15}$$

we only add one element to the new matrix. Therefore, at each stage, we need to add $t+2$ elements to the memory. The total memory requirements are $O(T^2)$. For most problems, the number of points $N$ is very large and $T << N$. Thus the storage requirements for MARK are much smaller that for methodologies based on the full $N \times N$ kernel matrix.

Another key to the efficiency of MARK-L is the equation solver used in Step 11. Our equation solver is based on a second-order optimization code based on a scaled conjugate gradient method where the Hessian matrix is made positive definite with a Levenberg-Marquardt term. Indeed, solving a symmetric system $Ax = y$ of equations is equivalent to minimizing $A'Ax - A'y + C$, where $C$ is an arbitrary constant. The scaled conjugate gradient method was utilized by Möller [13] as an alternate iterative higher-order learning algorithms for training neural networks, and it can be readily applied to general unconstrained optimization problems. This scaled conjugate gradient method for solving equations is a Krylov-type [11] method, that is fast, robust, and compact to code. It scales as $N^2$, rather than the $N^3$ which applies for most traditional equation solvers. Furthermore the equation solver can exploit the fact that we are solving a series of closely related equations in order to further enhance performance.

## 4. KERNEL COLUMN GENERATION

Another key aspect to the performance of MARK is the column generation method used in step 6 of MARK-L and MARK-S. Recall that the goal is to generate a column maximizing or approximately maximizing

$$|r^t K_i| \tag{16}$$

Returning to the underlying KKM with $Q$ types of kernels each centered about the $N$ data points, the column generation problem becomes

$$K_q(x, x_n) = \text{argmax}_{K_q,n} |\sum_{i=1}^{n} r_i^t K_q(x_i, x_n)| \tag{17}$$

Note that this is closely related to but not identical to choosing the column that most closely fits the residuals, i.e.

$$K_q(x, x_n) = \text{argmax}_{K_q,n} \sum_{i=1}^{n} (r_i^t - K_q(x_i, x_n))^2 \tag{18}$$

such as done in MART and additive basis models [9]. So MARK is not identical to these algorithms in this respect. One advantage of the kernel column selection problem is that it only requires the computation of a dot product. Consider the case of constructing a linear function for data stored in a relational database. Then the candidate kernel columns are the columns of the table containing the training data, and one can compute which column to generate via a simple SQL query.

The exact algorithm used to solve problem (17) depends on the kernel function. In this work we use regular RBF kernels and "weak" RBF kernels. Regular RBF kernels are assumed to be of the form

$$K(x, x_i) = e^{-\frac{\sum_{j=1}^{M}((x)_j - (x_i)_j)^2}{\sigma}} \tag{19}$$

where $x_i$ is one of the training data points, $(x_i)_j$ is the $j^{th}$ attribute of $x_i$ and $\sigma > 0$ is a parameter to be optimized. In the spirit of weak hypothesis in ensemble methods, we also constructed weak RBF kernels which are restricted to one attribute of the data set. So for example a weak RBF kernel defined based on the $j^{th}$ attribute and $i^{th}$ data point would be

$$K_j(x, x_i) = e^{-\frac{((x)_j - (x_i)_j)^2}{\sigma}} \tag{20}$$

The kernel $K_j$ is weak in the sense that if used in a regular SVM, it would not typically be sufficient for solving a problem alone. It is only as part of a HKM that such choice of kernels can be used. The benefit of using HKM composed of weak RBF kernels is that the kernels selected provide more interpretation than in the case of regular RBF kernels. A weak RBF kernel models inherently performs feature selection. Only a subset of the attributes will be used in the final model, and this is valuable information to the domain experts. In addition each weak RBF kernel has a somewhat clear meaning. For example if the attribute $j$ is size, the weak RBF kernel can be interpreted as identifying if the size is about that of instance $x_i$. Weak RBF kernels are only one possible type of weak kernel. The domain experts can introduce new kernels that make sense in a particular domain. Since there are no restriction on the matrix $K$ in MARK-L and MARK-S, these kernels need not obey Mercer's condition (see Section 3.) The only important thing is that one can define efficient algorithms to generate them.

For regular RBF kernels we use the following column generation algorithm. With an exhaustive search, the RBF column generation can be computationally expensive. To obtain better scalability, we try sampling uniformly from the data matrix $X$ and choose the best kernel column in the sample.

```
RBF Column Generation $(r, N')$
1. Construct a sample $S$ of data of size N'
2. For $n = 1, ..., N'$
     Call 1-D search and find $\sigma_n$ that optimizes (17) for fixed center $S_n$
3. Return kernel column $K_t$ for best found $S_n, \sigma_n$ pair
```

For weak regular RBF kernels we use the following column generation algorithm:

```
Weak RBF Column Generation $(r, N')$
1. Construct a sample $S$ of data of size N'
2. For $n = 1, ..., N'$
3.     For $j = 1, ..., M$
         Call 1-D search and find $\sigma_n$ that optimizes (17) for fixed center $(S_n)_j$
3. Return kernel column $K_t$ for best found $(S_n)_j, \sigma_n$ pair
```

There are many benefits to using weak RBF kernels. As discussed above, by analyzing the final weak kernel model, we can investigate the importance and influence of attributes by looking at the kernel columns added to the model and their associated weights. This leads to a way of non-linear feature selection. Another advantage of weak RBFs is that the number of parameters associated with a weak RBF kernel is dramatically reduced from M+1 for regular RBF kernels to 2 for weak RBF. Thus the storage and computational time required for testing a HKM based on weak RBF kernels can be dramatically reduced from that of HKM based on regular kernels. The computational costs of regular RBF and weak RBF column generation are both dependent on the choice of the sample size $N'$. The experimental results in the next section shows how efficient sampling can be in practice as well as comparison between regular and weak RBF kernels.

## 5. COMPUTATIONAL RESULTS

We perform a series of experiments to investigate the effectiveness of MARK-S and MARK-L on regression problems. In the next section, we examine the effect of parameters and variations of MARK. Then we look at generalization of MARK as compared to SVM and KRR algorithms. Finally we examine the scalability of MARK.

### 5.1 Behavior of MARK

We examine the effect of sampling on generalization with the column generation algorithm. It is important to know the performance of sampling since exhaustive search is computationally prohibitive.

Representative results for Boston Housing available from the UCI Machine Learning Repository [1] are presented. Boston Housing consists of 506 data points and 14 attributes. The results for 10-fold cross validation for normalized data are presented in Figures in the section 5.1. The statistic reported is $Q^2$ which is just the predicted mean squared error divided by the sample variance. Small values of $Q^2$ are better.

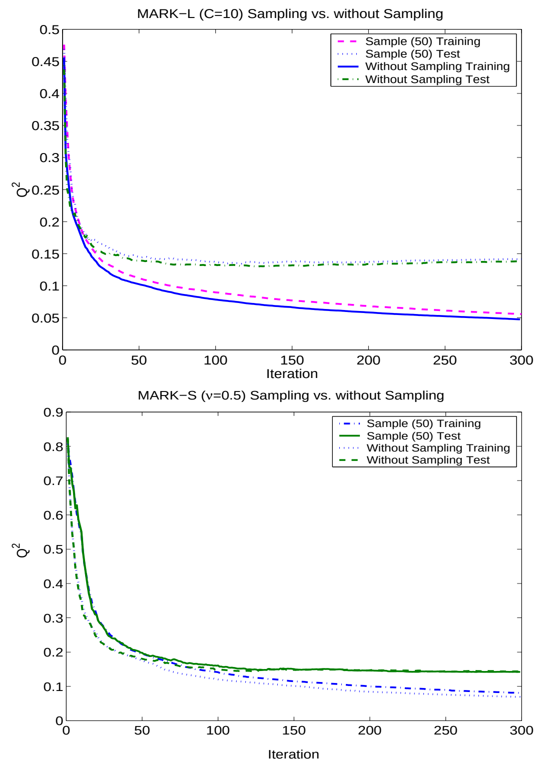


**Figure 1: Comparison between sampling (size = 50) and without sampling. For both cases the same value of regularization constant ($C = 10$) and weak RBF kernels were used. Both test and training learning curves are plotted. above: MARK-L, below: MARK-S.**

Figure 1 illustrates the effect of sampling on MARK-L and MARK-S. The sample size is set to 50. The kernel space is restricted to weak (column) kernels. As seen in the figure, the differences between sampling and without sampling are small for both MARK algorithms. Although the difference in hypothesis space size is extremely large, the weak learning algorithm succeeds in tuning the kernel parameters and makes use of available small sample size to fit the training examples. This observation is consistent with other kernel sampling methods such as in RSVM [8].

Next, we illustrate the difference between regular RBF kernels and weak RBF kernels. Sample results for MARK-L with $C = 10$ and sample size 50 are shown in Figure 2. Surprisingly, weak kernels converge much quicker than regular RBF kernels. This behavior comes from the fact that the importance of the data is not 'symmetric' in each dimension and, therefore, the assumption that there exists a best kernel width $\sigma$ for regular RBFs does not hold in this dataset. MARK algorithm with sampling gives a practical way to tune the width in each dimension so that we can make use of geometrical information inherent to the dataset. Weak RBF kernels inherently perform feature selection and this leads to faster learning speed. By choosing the attributes and adjusting the width of the kernel, MARK tunes the significance of each attribute individually.

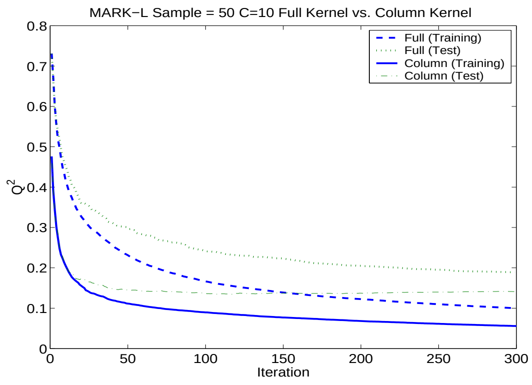
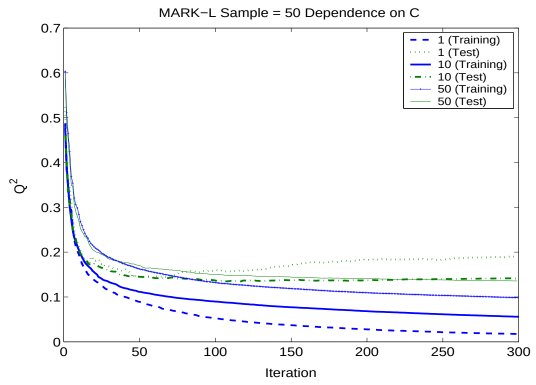


**Figure 2: Comparison between regular full kernel and weak column kernel.**

Next we examine the effect of the regularization parameters $C$ in MARK-L and $\nu$ in MARK-S. In Figure 3, learning curves are shown for each of MARK-L and MARK-S for various regularization parameter choices. Once again the sample size is set to 50 and weak RBF kernels are used. When the degree of regularization is small, for example $C = 1$ and $\nu = 1$, the test learning curve exhibits over-fitting. However, increasing regularization slows convergence. For example when $\nu = 0.1$ in Figure 3, the training error is high showing that learning is not yet finished. Shrinkage creates regularization by slowing down the learning rate. It is an implicit form of introducing flatness, but it is more difficult to tune compared to $C$ in MARK-L and results in HKM with more basis functions. Therefore, we prefer MARK-L to MARK-S. In general, MARK-L converges faster and gives stable generalization errors by explicitly controlling 'capacity' via $C$. MARK-L requires the solution of a system of equalities at each iteration and storage of the necessary matrix in order to compute $\alpha$ and $b$ at each iteration, but this added expense is justified. The best parameter that we obtain is $C = 10$ with MARK-L and this value will be used as a 'fixed policy' for the remainder of this paper. Better results may be obtained by tuning $C$.



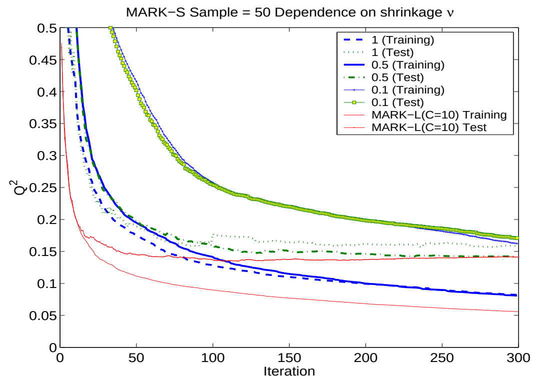


**Figure 3: Dependence on parameters. above: MARK-L, below: MARK-S.**

## 5.2 Benchmark Studies

We compare MARK with SVM and KRR on three datasets. For SVM, SVMTorch [3] is used. The three datasets are HIV, Boston Housing and Abalone. HIV is Quantitative Structure Activity Relationship (QSAR) data on reverse transcriptase inhibition of HIV generated as part of the DDASSL project at RPI (http://www.drugmining.com). HIV has 64 data points and 230 real attributes. Boston Housing has 506 data points and 13 attributes. Abalone has 4177 data points and 9 attributes. We note that HIV and the other two data sets have quite different shapes of the data matrix. Specifically, $M >> N$ for HIV and $M << N$ for others. As for HIV, since there are much less data points than attributes, we perform leave-one-out cross validation to obtain the test error estimate. For the other two datasets, a ten-fold cross validation is used for error estimation. All the computations are done on Windows 2000 machine with Pentium III 886MHz and 512MB RAM.

Based on the results in the previous section, MARK-L is used with weak RBF kernels, sample size 50, $C$ =10, and number of iterations $T = 100$. This fixed policy is not necessarily optimal for all datasets. For SVM and KRR, model and kernel parameters selection is automatically done by Pattern Search [14] using a validation set for each of the 10 folds. The size of validation set is set to 20% of the training set.

Leave-one-out cross validation results for HIV are presented in Table 1. The MSE for MARK-L is the lowest of the three approaches. We know results on HIV can be greatly improved by incorporating feature selection before running SVM or KRR. So one reason for the advantage of MARK-L may be that it inherently selects feature through the use of the weak RBF kernels. KRR used 63 support vectors since it is not a sparse method and there are 63 points in each training set. SVM requires only 27 full RBF kernels. MARK-L is set to always use 100 weak RBF kernels. The total number of parameters used by MARK-L can be found by the following argument; For each weak kernel we need to store two parameters, namely the center of kernel and the width $\sigma$, and the associated $\alpha_i$. In addition For $T$ weak kernels, therefore, the number of elements we store is $3T + 1 = 300 + 1$ (the 1 is for the threshold $b$). However, for SVMs, we cannot throw away any attributes of a support vector and only one parameter $\sigma$ is required therefore, the total number of information that has to be recorded is ((# support vectors)$\times M + 2$. In this case, $3T + 1 = 301$ and (# support vectors)$\times M + 2 = 6304$ (average). Therefore, the MARK model requires significantly less memory and is in some sense a simpler model. Specificaly, the ratio of storage required by SVM to MARK is 20.9. With a dataset like this, dimension reduction is necessary to obtain a good model. MARK with weak RBF kernels automatically performs feature selection, which is a key feature for success on QSAR type problems. In addition the reduction in model complexity produces an even greater reduction in computational costs in the testing phase.

| | MARK-L | SVMTorch | KRR |
|---|---|---|---|
| MSE | 0.333 | 0.353 | 0.567 |
| SV | 100 | 27.4 | 63 |
| Storage | 1 | 20.9 | 48.9 |

**Table 1: Leave-one-out test results for HIV. SV is the average number of support vectors or kernel columns used by each methods. Storage is the ratio of model storage required for the indicated method versus MARK-L**

Ten-fold cross validation results for Boston and Abalone are shown in Table 2 and 3. MARK performed comparably to SVMTorch on both datasets. KRR performs uniformly worse. Tuning of MARK parameters may further improve results. Pattern Search methodologies for tuning MARK are currently being tested. Moreover, as the size of dataset grows, storage requirements of the prediction model, in general, grows. The computational cost and storage requirements for reproducing the model for prediction unknown test data cannot be ignored. In SVM, if the number of support vector is $S$, the computational cost to produce a prediction is $O(M \times S)$. In MARK with weak kernels, it is just $O(1 \times T)$. This advantage cannot be ignored when dealing with a large dataset. On Abalone, a predictive SVM model requires $1375 \times 9 + 2$ units of information (4 bytes for a double precision variable). The factor '9' accounts for the number of attributes. Addition of '2' accounts for threshold $b$ and the kernel parameter $\sigma$. For MARK, it is $100 \times 3 + 1$ where '3' accounts for $\alpha$ and $\sigma$ for each center of the column kernel, '1' accounts for the threshold $b$. The ratio is $\frac{(1375\times9+2)}{(100\times3+1)} = 41$. Therefore, in this example, the memory requirement for MARK is about 41 times less than that for SVM. There would also be a corresponding decrease in classification costs. Similar ratios are provided for all the results in the table.

| | MARK-L | | SVMTorch | | KRR | |
|---|---|---|---|---|---|---|
| | Train | Test | Train | Test | Train | Test |
| MSE | 7.85 | 12.4 | 6.22 | 11.5 | 7.92 | 14.0 |
| Std | 6.80 | 7.0 | 2.25 | 6.95 | 7.79 | 6.86 |
| SV | 100 | | 256 | | 364 | |
| Storage | 1 | | 11.1 | | 15.7 | |
| Time | 6.0 | | 1.02 | | 20.25 | |

**Table 2: Results for Boston Housing**

| | MARK-L | | SVMTorch | | KRR | |
|---|---|---|---|---|---|---|
| | Train | Test | Train | Test | Train | Test |
| MSE | 4.30 | 4.65 | 4.26 | 4.50 | 6.17 | 6.66 |
| Std | 0.052 | 0.504 | 0.10 | 0.486 | 3.50 | 4.36 |
| SV | 100 | | 1375 | | 3579 | |
| Storage | 1 | | 41.1 | | 105.5 | |
| Time | 57.0 | | 22.7 | | $5.7 \times 10^5$ | |

**Table 3: Results for Abalone**

## 5.3 Scalability

In the final experiment we investigate the scalability of MARK-L on a large dataset, the DELVE Housing8H dataset with 22,784 data points and 8 attributes. We compare the computational time with SVMTorch version 2.2. In this experiment, it does not make sense to use $T = 100$ because as the number of data points increases it is reasonable for the capacity of the learning function to increase as well. Therefore, we use a sampling of 5,000 data points to find a baseline MSE and use this value as a stopping criterion for MARK-L. For the SVM, we use the fixed policy described in [14]. MARK-L exhibits near linear scaling.

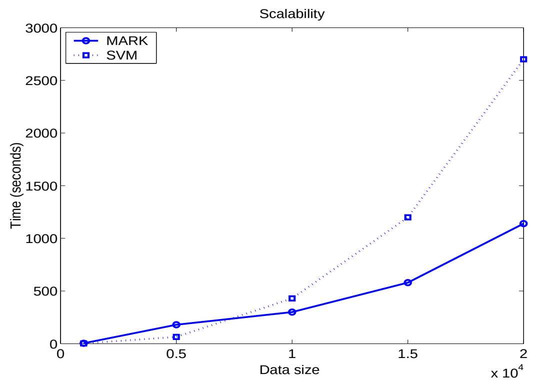


**Figure 4: Scalability of MARK-L compared with SVM on DELVE Housing8H.**

## 6. CONCLUSIONS

This work was motivated by our long term goal to construct kernel methods that can more easily incorporate and extract domain knowledge. Within an application domain there are many notions of distance. HKM allows the domain experts to incorporate many different notion of distance into a kernel based model. We proposed the MARK algorithm for constructing inference models based on these heterogeneous families of kernels. Motivated by ensemble and additive model methods, the MARK algorithm utilizes a coordinate descent approach to optimize the heterogeneous kernel models. Analogous to base learning algorithms in ensemble methods, kernel columns from the heterogeneous kernel matrix are generated on the fly. Since many kernels can be combined, we can utilize weak kernels such as the weak RBF kernel examined in this work. Computational results showed that MARK with weak kernels produced models of similar quality to SVM but with dramatically less storage requirements. One benefit of MARK is that the method inherently performs feature selection, an important property for problems such a QSAR/drug discovery and micro-array gene expressions where there are many attributes relative to the number of points. In addition, the weak kernels are in some sense interpretable. Therefore we are investigating approaches for interpreting and visualizing the resulting HKM.

## 7. REFERENCES

[1] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[2] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1/3):131, 2002.

[3] R. Collobert and S. Bengio. Support vector machines for large-scale regression problems. *IDIAP-RR-00-17*, 2000.

[4] N. Cristianini and J. Shawe-Taylor. An introduction to support vector machines and other kernel-based methods, 2000.

[5] N. Duffy and D. Helmbold. Leveraging for regression. In *Proc. COLT*, pages 208–219, 2000.

[6] Y. Freund and R. Shapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[7] J. Friedman. Greedy function approximation. *Technical report, Department of Statistics, Stanford University*, February 1999.

[8] G. Fung and O. Mangasarian. Proximal support vector classifiers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 77–86. ACM, 2001.

[9] T. Hastie, R. Tibshairani, and J. Friedman. *The elements of statistical Learning*. Springer, 2001.

[10] T. Hastie and R. Tibshirani. Generalized additive models. *Statistical Science*, 1:297–318, 1986.

[11] C. F. Ipsen and C. D. Meyer. The idea behind krylov methods. *Amer. Math. Monthly*, 105(10):889–99, 1998.

[12] L. Mason, P. Bartlett, J. Baxter, and M. Frean. Functional gradient techniques for combining hypotheses. In B. Schölkopf, A. Smola, P. Bartlett, and D. S. ans, editors, *Advances in Large Margin Classifiers*. MIT Press, 2000.

[13] M. F. Möller. A scaled conjugate gradient algorithm for supervised learning. *Neural Networks*, 6:525–533, 1993.

[14] M. Momma and K. P. Bennett. A pattern search method for model selection of support vector regression. In *Proceedings of the Second SIAM International Conference on Data Mining*. SIAM, 2002. to appear.

[15] G. Rätsch. *Robust Boosting via Convex Optimization: Theory and Applications*. PhD thesis, University of Potsdam, Department of Computer Science, 2002.

[16] G. Rätsch, A. Demiriz, and K. Bennett. Sparse regression ensembles in infinite and finite hypothesis spaces. *Machine Learning*, 48((1-3)):193–221, February 2002.

[17] G. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proc. 15th International Conf. on Machine Learning*, pages 515–521. Morgan Kaufmann, San Francisco, CA, 1998.

[18] A. Smola, Bartlett, B. Schölkopf, and D. Schuurmans. *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 2000.

[19] J. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[20] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998.