Some Greedy Algorithms for Sparse Nonlinear Regression

Prasanth B. Nair P.B.NAIR@SOTON.AC.UK

Arindam Choudhury A.CHOUDHURY@SOTON.AC.UK

Andy J. Keane ANDY.KEANE@SOTON.AC.UK

Computational Engineering and Design Center, School of Engineering Sciences, University of Southampton, Highfield, Southampton SO17 1BJ, United Kingdom.

Abstract

We present deterministic greedy algorithms for building sparse nonlinear regression models from observational data. Our objective is to develop efficient numerical schemes for reducing the training and runtime complexities of nonlinear regression techniques applied to massive datasets. In the spirit of Natarajan's greedy algorithm (Natarajan, 1995), we iteratively minimize a loss function subject to a specified constraint on the degree of sparsity required of the final model or an upper bound on the empirical error. We discuss various greedy criteria for basis selection and numerical schemes for improving the robustness and computational efficiency. Subsequently, algorithms based on residual minimization and thin incremental QR factorization updating are presented for constructing sparse regularization networks. Experimental results on some benchmark problems are presented to demonstrate the competitiveness of the algorithms developed in this paper.

1. Introduction

We consider the problem of constructing a regression model using observational data $\mathcal{D} := (x_i, y(x_i)), i = 1, 2, \ldots, m$, where $x_i \in \mathbb{R}^p$ denotes the input vector and $y \in \mathbb{R}$ denotes the target to be approximated. In particular, we focus on regression models of the form $\hat{y}(x) = \sum_{j=1}^{n} \alpha_j k(x, x_j^*)$, where $\{x_j^*\}_{j=1}^{n} \subseteq \mathcal{D}$. Further, $k(x, x_i)$ is a Mercer kernel (Vapnik, 1995), i.e., the Gram matrix $K \in \mathbb{R}^{m \times m}$ with elements $K_{ij} = k(x_i, x_j)$ is symmetric positive definite (SPD). We compactly denote the weight vector as $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_m\} \in \mathbb{R}^m$, and the vector of targets as $y = \{y(x_1), y(x_2), \ldots, y(x_m)\} \in \mathbb{R}^m$.

The structure of the regression model considered here is typical of that encountered in regularization networks and support vector regression (Evgeniou, Pontil and Poggio, 2000). In the absence of regularization, the weight vector can be computed by solving $K\alpha = y$. As a consequence the resulting model behaves as an interpolator. In contrast, when the targets are corrupted by noise, $y = K\alpha + \eta$, where $\eta \in \mathbb{R}^m$ denotes noise which is typically assumed to be $\mathcal{N}(0, \sigma^2)$. Hence, one needs to solve $(K + \sigma^2 I)\alpha = y$ to compute the regularized weight vector. Note that computation of the weight vector for both cases would require $\mathcal{O}(m^2)$ memory and $\mathcal{O}(m^3)$ operations, which would clearly be prohibitive for massive datasets.

1.1 Sparse Approximations

One way to reduce the memory requirements and computational cost would be to solve the overdetermined least-squares problem: $\min ||K^n \alpha^n - y||_2$, where $K^n \in \mathbb{R}^{m \times n}$ is a rectangular matrix formed by choosing a subset of the m columns of the original Gram matrix and $\alpha^n \in \mathbb{R}^n$ denotes the truncated weight vector. Note that since K is SPD, K^n has full column rank. This approach intrinsically leads to a *sparse* regression model since $n < m$ and hence only a subset of the training data is employed in the final regression model. In addition, this approach requires only $\mathcal{O}(mn)$ memory and $\mathcal{O}(n^3)$ operations for computing the weight vector. Further, the sparse model will require only $\mathcal{O}(n)$ operations for predicting the target at a testing point.

Since the empirical/training error of the sparse approximation becomes $||K^n \alpha^n - y||_2 = \epsilon > 0$ for $n << m$, such a regression model may also be referred to as a sparse approximate interpolator. Recently, Natarajan (1999) presented a theoretical justification for regularization via sparse approximate interpolation. It was shown that interpolation error (ϵ) and noise in the target vector y will tend to cancel out if ϵ is chosen *a priori* to be the noise intensity. Natarajan's sparse approximation problem can be formulated as: find a sparse vector $\alpha \in \mathbb{R}^m$ such that $||K\alpha - y|| \leq \epsilon$. Alternatively, this problem may be interpreted as a subset selection problem of the form: find n columns of the matrix K such that $\min_{\alpha^n} ||K^n \alpha^n - y||_2 \leq \epsilon$.

Clearly, choosing n columns out of m possible choices involves a combinatorial search over an ${}^m\mathrm{C}_n$ space. As shown by Natarajan (1995), this problem is NP-hard, and we have to resort to suboptimal search schemes to ensure computational efficiency. Note that the problem statement considered here appears in the literature in different guises, for example, basis selection methods, basis pursuit, matching pursuits (Chen, Donoho, and Saunders, 1998), and sparse approximate inverse computation (Chow and Saad, 1997; Grote and Huckle, 1997).

1.2 Outline of the Paper

In this paper, we present some greedy algorithms for constructing sparse nonlinear regression models. In particular, our focus is on sparsifying regularization networks based on Mercer kernels for function approximation. We examine the computational properties of existing greedy algorithms in the domain of signal processing and preconditioned iterative numerical methods. Drawing on developments in these fields, we derive iterative algorithms based on residual error minimization and thin incremental QR factorization for sparse nonlinear regression. We show that the present algorithms are computationally more efficient and require significantly less memory than Natarajan's algorithm. Numerical studies on some popular benchmark problems also suggest that our algorithms lead to regression models with good generalization abilities.

This paper is organized as follows: In section 2, we present some background material on greedy algorithms. Section 3 outlines various considerations involved in developing robust greedy algorithms with modest memory requirements and low computational cost. Section 4 presents a residual minimization technique for updating the weight vector during the iterations of a greedy algorithm. In section 5, we describe a greedy algorithm for incremental QR factorization. Section 6 presents experimental investigations on a synthetic dataset as well as some real-world datasets, and Section 7 concludes the paper.

2. Preliminaries

The greedy algorithms in the literature can be broadly classified into sequential forward greedy algorithms (Mallat and Zhang, 1993; Natarajan, 1995; Smola and Bartlett, 2001; Grote and Huckle, 1997), backward greedy algorithms (Couvreur and Bresler, 1999), and mathematical programming approaches (Chen, Donoho, and Saunders 1998; Girosi, 1998). In general, sequential forward algorithms are computationally cheap and tend to have modest memory requirements. In contrast, backward greedy algorithms are computationally much more expensive since the full-order Gram matrix is factorized prior to iteratively annihilating columns which lead to minimal increment in the residual error (Couvreur and Bresler, 1999). From a theoretical point of view, backward algorithms have provable convergence properties. In contrast, forward algorithms offer no such known guarantees. Mathematical programming approaches to sparse approximation attempt to reduce the $\mathcal{L}_0$ norm of the weight vector by minimizing an approximation to it; see, for example, Girosi (1998) and Chen, Donoho, and Saunders (1998). The computational cost associated with this class of algorithms also tends to be much higher than sequential forward greedy approaches.

In the present paper, we restrict our discussion to sequential forward greedy algorithms. We present below a brief overview of one such representative sequential forward greedy scheme, viz. Natarajan's algorithm. A comparison study by Adler, Rao and Kreutz-Delgado (1996) has shown that this algorithm generally performs better than other algorithms in the literature. Hence, we will benchmark the numerical schemes developed in the present research against Natarajan's algorithm.

The following notation has been used throughout this paper; we denote the inner product between two vectors as $(x, y) = x^T y$. $e_j \in \mathbb{R}^m$ is used to denote the jth unit vector. We employ "colon" notation of the form $K(:, I) \in \mathbb{R}^{m \times p}$ to denote the matrix formed using column indices chosen from the set I of cardinality p. We will refer to the i_pth column of the Gram matrix $K \in \mathbb{R}^{m \times m}$ by $k_{i_p} \in \mathbb{R}^m$. α_j will be used to refer to the jth element of the vector α. Further, the superscript ^{p} will be used to denote the iteration index.

2.1 Natarajan's Algorithm

Natarajan's algorithm (Natarajan, 1995) consists of recursively choosing and modifying the columns of K in an iterative fashion so as to maximally reduce the $\mathcal{L}_2$ norm of the residual error vector $r = y - K\alpha$. Let $K^{p-1} \in \mathbb{R}^{m \times (p-1)}$ denote the matrix of $p-1$ columns chosen till the $(p-1)$th iteration, and let I^{p-1} represent the set of corresponding column indices in K^{p-1}. The pth iteration then involves choosing a previously unselected column k_{i_p} from K such that $||r||_2$ is maximally reduced. The first step of Natarajan's algorithm involves normalizing the columns of K. Let $\tilde{K}$ denote the normalized Gram matrix. The steps involved at the pth iteration are shown below:

$$
\begin{aligned}
i_p &= arg \max_{j \notin I^{p-1}} \mid (\tilde{k}_j^{p-1}, r^{p-1}) \mid && (1)\\
I^p &\leftarrow [I^{p-1}, i_p] && (2)\\
r^p &\leftarrow r^{p-1} - (\tilde{k}_{i_p}^{p-1}, r^{p-1})\tilde{k}_{i_p}^{p-1} && (3)\\
\tilde{k}_j^p &\leftarrow \tilde{k}_j^{p-1} - (\tilde{k}_{i_p}^{p-1}, \tilde{k}_j^{p-1})\tilde{k}_{i_p}^{p-1}, \quad \forall\, j \notin I^{p-1} && (4)\\
\tilde{k}_j^p &\leftarrow \frac{\tilde{k}_j^p}{||\tilde{k}_j^p||_2} \quad \forall\, j \notin I^{p-1} && (5)
\end{aligned}
$$

Note that the residual error vector r is initialized to y in the first iteration. The final weight vector $\alpha^n \in \mathbb{R}^n$ after n iterations can be computed by solving the least squares problem min $\parallel K^n\alpha^n - y \parallel_2$, where $K^n \in \mathbb{R}^{m\times n}$ denotes the rectangular Gram matrix formed using the indices in I^n.

It can be noted from (1) that the index selection criterion is applied only to previously unselected columns. Hence, this scheme does not suffer from the recycling problem encountered in greedy algorithms such as basis matching pursuit (Mallat and Zhang, 1993) and the scheme of Schaback and Wendland (2000). Hence, Natarajan's algorithm is also referred to as order recursive matching pursuit (ORMP).

Further, the following theorem guarantees a sparse weight vector at the end of the ORMP iterations when an upper bound ϵ on the norm of the residual is specified as the termination criteria.

Theorem 1[Natarajan, 1995]: *The number of non-zero elements in the weight vector $\alpha \in \mathbb{R}^m$ is at most*

$$r \leq [18Opt(\frac{\epsilon}{2})\parallel K^{-1} \parallel_2^2 \ln(\frac{\parallel y \parallel_2}{\epsilon})]$$

where $Opt(\frac{\epsilon}{2})$ denotes the fewest number of non zero entries over solutions that satisfy $\parallel K\alpha - y \parallel_2 \leq \frac{\epsilon}{2}$.

The major computational cost incurred in ORMP is in step (1) and the column update in step (4), which has to be undertaken for all the previously unselected columns at each iteration,[1] It can be noted that $\mathcal{O}(m^2)$ operations are required at each iteration. For massive datasets, this could significantly increase the training time. Further, the ORMP algorithm requires the computation and storage of the full-order Gram matrix. In contrast, the techniques presented in this paper have much smaller memory requirements, and are generally faster while achieving comparable (or better) generalization performances.

[1]Note that Natarajan's algorithm may be interpreted as a QR factorization scheme based on the modified Gram Schmidt algorithm, in which the column pivot is chosen greedily with respect to the vector y.

3. Structure of Greedy Algorithms

In this section, we present an overview of the structure of greedy algorithms developed in this paper. A fundamental component of a greedy algorithm is the choice of criterion employed to select a new column at each iteration. Once a new column is chosen, it is necessary to update the weight vector α and the residual error vector, $r = y - K\alpha$. Further, to ensure robustness, we also need to allow the flexibility of dropping a previously chosen column. We discuss next various considerations involved in designing a computationally efficient greedy algorithm.

3.1 Criteria for Basis Selection

This crucial step involves the use of a criterion for deciding which new column should be chosen at the current iteration. Clearly, it would be preferable to choose the column which leads to the greatest reduction in the residual error. One way to achieve this would be to examine the current residual (r), and search for the entry with highest absolute value. Assuming that the residual error is implicitly computed at each iteration, this criteria would incur negligible computational cost. Further, to implement this criteria, one only needs to store n columns of the Gram matrix, where n is the maximum number of iterations.

In line with Natarajan's algorithm, an alternative approach would be to work with the residual corresponding to the normal equations[2], i.e., $K^TK\boldsymbol{\alpha} = K^Ty$. However, this involves $\mathcal{O}(m^2)$ operations to compute the normal residual K^Tr, as well as the requirement of computing and storing the full-order Gram matrix.

A different criterion was proposed by Huckle and Grote (1997) in the context of constructing sparse approximate inverse preconditioners. They suggested that by solving a sequence of one-dimensional minimization problems (over the columns not yet selected), the column which leads to greatest reduction in the residual can be chosen. To illustrate this, let I^{p-1} denote the indices of the columns chosen at the (p-1)th iteration. A new column can then be selected at iteration p by solving (m-p+1) one-dimensional minimization prob-

[2]Note that this is essential only when the coefficient matrix is indefinite. However, since the present research is restricted to Gram matrices based on Mercer kernels (i.e., K is SPD), this is not strictly required.

lems of the form

$$\min_{\mu_j, j \notin I^{p-1}} \| r^{p-1} + \mu_j K e_j \|_2 \quad (6)$$

The solution of this minimization problem turns out to be

$$\mu_j = -\frac{(r^{p-1}, Ke_j)}{(Ke_j, Ke_j)} \quad (7)$$

For each j, the $\mathcal{L}_2$ norm of the new residual $r^{p-1} + \mu_j K e_j$ can be written as

$$\rho_j^2 = \| r^{p-1} \|_2^2 - \frac{(r, Ke_j)^2}{(Ke_j, Ke_j)} \quad (8)$$

Note that solving all the $(m - p + 1)$ minimization problems would require the computation and storage of the full-order Gram matrix. Recently, Smola and Schölkopf (2000) proposed a general probabilistic approach for sparse approximation. The success of their approach (cf. Smola and Bartlett, 2001) suggests that, in practice, by solving around 60 one-dimensional minimization problems for randomly chosen column indices (where, column index $j \notin I^{p-1}$), columns which are probabilistically in the top 95 percentile can be chosen. Such a probabilistic approach may hence lead to a significant reduction in the number of one-dimensional minimization problems to be solved. However, in the present paper, we pursue the simpler criterion which involves searching for the index with the largest absolute value in r.

3.2 Updating α and r

Once a new column is selected, we need to update the weight vector α and the residual error vector r. Further, it is desirable to do this step with minimal computational effort.

Smola and Bartlett (2001) update the weight vector by explicitly updating the inverse of $E(:, I^p)^T K E(:, I^p)$ when an additional column is added at each iteration. However, such explicit updates when computed frequently could lead to numerical instabilities due to round-off error accumulations. In the present research, we consider two alternative schemes which are expected to be numerically more stable. The first scheme described in Section 4 is an iterative scheme based on residual error minimization. The second scheme presented in Section 5 uses an thin incremental QR factorization update scheme. Both these schemes only incur $\mathcal{O}(mn)$ operations at each iteration of the greedy algorithm.

ALGORITHM 1: Exchange Algorithm	
1.	$s \leftarrow arg \min_{j \in I^p} \lvert r_j^p \rvert$
2.	$l \leftarrow arg \max_{j \notin I^p} \lvert r_j^p \rvert$
3.	$\gamma \leftarrow \frac{(r^p + \alpha_s^p K e_s, K e_l)}{(K e_l, K e_l)}$
4.	$\tilde{r} \leftarrow r^p + \alpha_s^p K e_s - \gamma K e_l$
5.	**If** $\lVert \tilde{r} \rVert < \lVert r^p \rVert$ then
	$\alpha_s^p \leftarrow 0$
	$\alpha_l^p \leftarrow \gamma$
	$r^p \leftarrow \tilde{r}$
	$I^p \leftarrow [I^p \setminus s, l]$
	End if

3.3 Exchange Algorithm

Most forward greedy algorithms in the literature do not allow for the possibility of annihilating a column once it has been chosen. Hence, for some problems, a forward greedy algorithm which makes a myopic choice at an early iteration may end up choosing an exorbitant number of additional columns to compensate for the previous choice. To circumvent this possibility, we include a step which allows for the flexibility of dropping a column based on the progress of the iterations. There are many possibilities for achieving this. For example, one could integrate backward (Couvreur and Bresler, 1999) and forward greedy algorithms to combine the robustness of the former and the computationally efficiency of the latter approach. However, here we pursue a simpler approach motivated by the algorithm presented by Chow and Saad (1997) for computing sparse approximate inverses.

The basic idea is to drop the index for which $|r|$ is lowest (say $s \in I^p$) in favor of a previously unselected column index with the largest value in $|r|$ (say $l \notin I^p$), if this leads to a reduction in the current residual. To check whether this exchange strategy will lead to a lower residual, we need to solve the one-dimensional minimization problem:

$$\min_{\gamma} \| y - K(\alpha^p - \alpha_s^p e_s + \gamma e_l) \|_2 \quad (9)$$

The solution of this minimization problem is given by

$$\gamma = \frac{(r^p + \alpha_s^p K e_s, K e_l)}{(K e_l, K e_l)} \quad (10)$$

where α^p and r^p denotes the weight vector and residual error at iteration p, respectively.

The steps of the exchange algorithm are summarized in Algorithm 1. Note that the operation $I^p \setminus s$ indicates the process of removing the index s from the set I^p. It can be seen that the exchange step will cost $\mathcal{O}(m)$ additional operations at each iteration of the greedy algorithm. A similar exchange strategy can be coupled to a thin incremental QR factorization scheme as discussed in Section 5.

4. Minimal Residual Iterations

This section presents a minimal residual iteration scheme for updating α and r. The basic idea of the minimal residual iteration scheme (Chow and Saad, 1997) is to zero-out many terms of the search direction to ensure sparse solutions. The steps of this iterative scheme are summarized in Algorithm 2. Let r^p and α^p denote the residual error and the weight vector, respectively, at the end of the pth iteration. At the next iteration, we use the search direction d, which is the residual error vector constrained to have the same sparsity pattern as α^p plus a new entry (see steps 2-4). This new entry is chosen to be the largest component in $|r^p|$. Minimization is carried out using the steplength computed in step 6. This guarantees that $||r^{p+1}|| \leq ||r^p||$. The residual error and weight vector are updated using this steplength in steps 7-8. In the last step, we apply the exchange strategy described earlier in Algorithm 1. We henceforth refer to the minimal residual iteration scheme with exchange as MRX.

Note that d is a sparse vector with only p non-zero entries. Hence, computation of Kd in step 5 of Algorithm 2 requires only p columns of K. Hence, the MRX algorithm only requires a maximum of $\mathcal{O}(mn)$ memory. Further, the computational complexity at each iteration is $\mathcal{O}(mp)$.

ALGORITHM 2: Minimal Residual Iterations

Input: maximum allowable sparsity n.
Set $p = 0$, $\alpha^0 = 0$, $r^0 = y$, and $I^p = [\,]$
while $p < n$, **do**
1. $p \leftarrow p + 1$.
2. Find $i_p = arg \max_{j \notin I^{p-1}} |r_j^{p-1}|$
3. $I^p \leftarrow [I^{p-1}, i_p]$
4. $d_j \leftarrow r_j^{p-1} \quad \forall j \in I^p$ and $d_j \leftarrow 0 \quad \forall j \notin I^p$.
5. $q \leftarrow Kd$
6. $\beta \leftarrow (r^{p-1}, q)/(q, q)$
7. $r^p \leftarrow r^{p-1} - \beta q$
8. $\alpha^p \leftarrow \alpha^{p-1} + \beta d$
9. Apply Algorithm 1

5. QR Factorization Update Scheme

As an alternative to MRX, we now present a thin incremental QR scheme for updating α and r, when a new column of the Gram matrix is added at each iteration. As shown below, this enables the residual error vector to be readily updated during the iterations of the greedy algorithm. The sequence of steps at the pth iteration are summarized below:

$$i_p = arg \max_{j \notin I^{p-1}} | r_j^{p-1} | \tag{11}$$

$$I^p \leftarrow [I^{p-1}, i_p] \tag{12}$$

$$q^p \leftarrow Orth[Q^{p-1}, k_{i_p}] \quad \in \mathbb{R}^m \tag{13}$$

$$Q^p \leftarrow [Q^{p-1}, q^p] \quad \in \mathbb{R}^{m \times p} \tag{14}$$

$$R^p \leftarrow \begin{bmatrix} R^{p-1} & (Q^{p-1})^T k_{i_p} \\ 0 & (q^p, k_{i_p}) \end{bmatrix} \in \mathbb{R}^{p \times p} \tag{15}$$

$$r^p \leftarrow r^{p-1} - (q^p, y) q^p \tag{16}$$

where $q^p = Orth[Q^{p-1}, k_{i_p}]$ stores the result of orthogonalizing the new column vector k_{i_p} with respect to the columns of the orthogonal matrix $Q^{p-1} \in \mathbb{R}^{m \times (p-1)}$. We use the modified Gram Schmidt algorithm for orthogonalizing the new column. In contrast to Natarajan's algorithm presented in section 2.1, the present approach does not require explicit storage of all the columns of the Gram matrix K. Instead, one needs to compute only the required column(s) at runtime. Further, the stepwise QR update leads to a very simple update formula for the residual (16). Finally note that there is no explicit solution update stage. The solution may be obtained in one shot once the stopping criterion is met. Assuming that the scheme terminates at the nth iteration, the weight vector can then be computed by solving the upper triangular system of equations

$$R^n \alpha^n = (Q^n)^T y \tag{17}$$

where $R^n \in \mathbb{R}^{n \times n}$ and $Q^n \in \mathbb{R}^{m \times n}$. From a computational aspect, the QR-based update scheme requires $\mathcal{O}(mp)$ operations at the pth iteration. Note that our QR update scheme is expected to be much faster than Natarajan's algorithm, since we update a thin QR factorization at each iteration rather than updating all the unselected columns.

ALGORITHM 3: QR-based Exchange Strategy

If ($\| \tilde{r} \| < \| r \|$) then
  Downdate Q^p and R^p at the sth column:
    $R^p \leftarrow G_{p-1}^T \ldots G_s^T R^p(:, I^p \setminus s)$
    $Q^p \leftarrow Q^p G_s \ldots G_{p-1}$, where G_i denotes a rotation in planes $i, i+1$ for $i = s : p-1$.
  Update Q^p and R^p at the lth column using equations (13-15).
  Modify r^p using the updated factors.
End if

The incremental QR update scheme can be modified to

p	n	Methods		
		MRX	QR	ORMP
5	200	0.532± 0.094	0.483± 0.059	0.483± 0.059
	300	0.452± 0.081	0.405± 0.032	0.405± 0.033
	400	0.409± 0.096	0.349± 0.021	0.349± 0.021
10	200	0.336± 0.006	0.335± 0.006	0.335± 0.006
	300	0.299± 0.006	0.298± 0.006	0.298± 0.006
	400	0.266± 0.005	0.265± 0.005	0.265± 0.005

Table 1. Comparison of statistics of residuals for 100 realizations of the synthetic datasets generated using F^*.

incorporate an exchange algorithm in the spirit of section 3.3. However, this involves applying a QR downdate followed by an update operation. Specifically, at the pth iteration, one drops the column for which the corresponding residual is the lowest (say s) in favor of a column $\notin I^p$ (say l). Thus we need only modify step (5) in Algorithm 1. The update always occurs at the tail of the QR factorization and is essentially described by Eqns. 13-15. The downdate at the sth position can be achieved by applying a sequence of Given's rotations to annihilate the subdiagonal terms of the upper Hessenberg matrix ${Q^p}^T K(:, I^p \setminus s)$, where $\setminus s$ denotes the removal of the column index s from the set of column indices in I^p. Therefore, step 5 of Algorithm 1 is modified as shown in Algorithm 3.

6. Experimental Investigations

The algorithms presented in this paper were implemented as a library of FORTRAN 77 subroutines which make extensive use of machine optimized BLAS routines. All the numerical experiments reported here were conducted on a single R10000 processor of an SGI Origin 2000 machine with 4.6 Gb main memory. Further note that unlike ORMP, the present algorithms do not require the explicit *a priori* computation and storage of the full Gram matrix K.

The algorithms developed in this work are first tested on a synthetic dataset generated according to (Friedman, 1999). Subsequently, a detailed comparison study is performed on two real world datasets viz. Boston and Abalone available from the UCI machine learning repository.

For all experimental investigations, the elements of the Gram matrix are constructed using a Gaussian kernel of the form $k(x_1, x_2) = \exp(-\frac{(x_1-x_2)^2}{2\omega^2})$. We set $2\omega^2$ to 0.01, 3.9 and 10.0 for the synthetic, Boston and Abalone datasets, respectively. Further, we work with the regularized Gram matrix $K \leftarrow K + \sigma^2 I$, where σ^2 is chosen to be 0.01 for both the synthetic and Boston datasets and 0.1 for Abalone.

The synthetic dataset is generated using a linear com-

Subset size (n)	Methods		
	MRX	QR	ORMP
50	14.13± 8.07	13.68± 7.65	12.17± 7.01
100	12.19± 7.22	11.19± 6.51	10.43± 6.47
150	11.08± 6.50	10.48± 6.01	10.05± 7.15
200	8.87± 5.67	8.35± 5.67	10.13± 7.02

Table 2. MSE comparison of MRX, QR & ORMP for the Boston dataset for varying subset sizes.

bination of 20 basis functions (Friedman, 1999), i.e, $F^*(x) = \sum_{l=1}^{20} a_l \mathbf{g}_l(x) + \mathcal{N}(0, \sigma^2)$, where $x \in \mathbb{R}^p$ and the scalar a_ls are drawn from a uniform distribution $[0, 1]$. Here $\mathbf{g}_l(x) = \exp\{-(x-\mu_l)^T \Sigma^{-1}(x-\mu_l)\}$, where $\mu_l \in \mathbb{R}^p$ is drawn from a uniform distribution $[0, 1]^p$ and $\Sigma \in \mathbb{R}^{p\times p}$ is a randomly generated SPD matrix.

We compare the performance of the minimal residual iteration (MRX) scheme and the incremental QR factorization algorithm with Natarajan's algorithm (ORMP). The convergence of the algorithms are analyzed by examining the normalized residual error $||y - K\alpha||_2/||y||_2$.

Table 1. summarizes the results of applying the above techniques to 100 realizations each of F^* for dimensionalities p=5 and p=10, respectively, over a range of subset sizes (n). For each realization, a randomly generated training dataset of 2000 points is used. It appears that the incremental QR and ORMP algorithms are superior to MRX on an average, for either dimensionality (p). However, the differences in performance between the incremental QR-based technique and ORMP are clearly statistically insignificant.

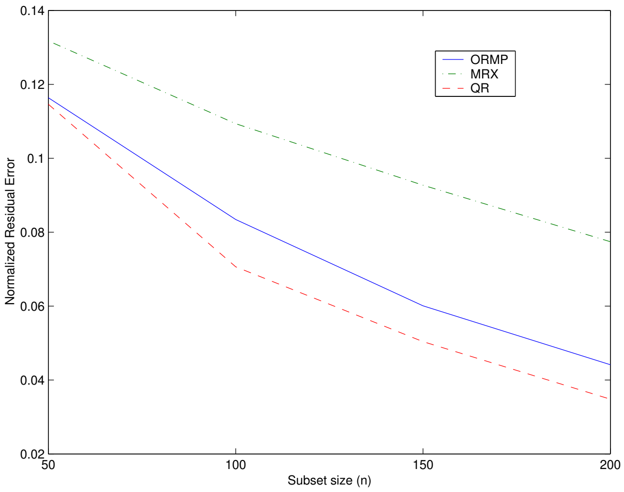

Figure 1. Convergence of the averaged residual error using greedy algorithms on the Boston dataset.

Fig. 1 and Table. 2 show the convergence trends of the residual error and the mean square testing error, respectively, as the subset size is increased, for the

Subset size (n)	Methods		
	MRX	QR	ORMP
50	5.04± 0.30	4.83± 0.29	4.92± 0.29
100	4.71± 0.19	4.66± 0.30	4.66± 0.20
150	4.66± 0.29	4.65± 0.28	4.63± 0.25
200	4.52± 0.17	4.51± 0.27	4.45± 0.18
250	4.60± 0.25	4.54± 0.34	4.57± 0.23
300	4.59± 0.24	4.55± 0.21	4.57± 0.25
350	4.68± 0.29	4.53± 0.29	4.64± 0.31

Table 3. MSE Comparison of MRX, QR & ORMP for the Abalone dataset for varying subset sizes.

Boston dataset. The trends shown are averaged over 100 random splits of the mother data into a training set of 481 and a testing set of 25 instances. Clearly, the incremental QR factorization update technique succeeds in reducing the residual error the most, while still managing an MSE performance comparable to that of ORMP. This observation along with the attendant computational and storage benefits makes the incremental QR-based technique the algorithm of choice for this dataset. Also note that although MRX loses out in the race for reducing the residual error, its MSE performance is not significantly worse off than ORMP (and QR). For n=200, it actually performs better than ORMP (see Table 2.).

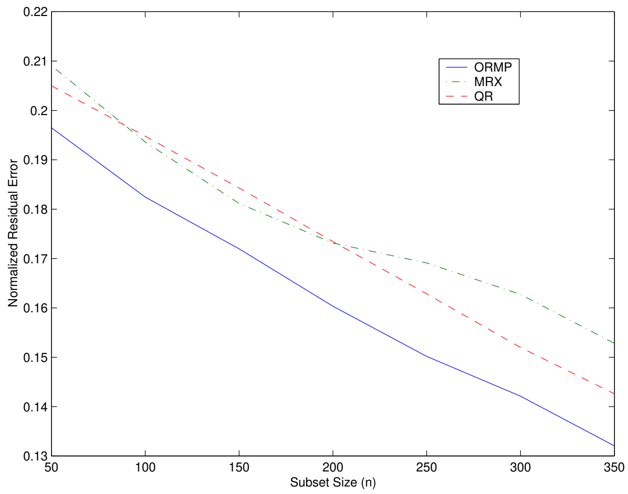

Figure 2. Convergence of the averaged residual error using greedy algorithms on the Abalone dataset.

From the results, it can be noted that the best test performance obtained using the incremental QR-based algorithm is 8.3 mean square error with standard deviation 5.7. This is comparable with the best performance achieved by a support vector machine (mean square error 8.7 and standard deviation 6.8) on this dataset; see Table 3 in Schölkopf, Smola, Williamson, and Bartlett (2000). Further, the QR-based algorithm only required around 0.8 seconds to compute the weight vector at the specified sparsity level of 200. The training speed is hence higher than the state-of-the-art support vector machine formulation of Mangasarian and Musicant (2000), which takes around 36 seconds for this dataset (refer Table 2. in that paper).

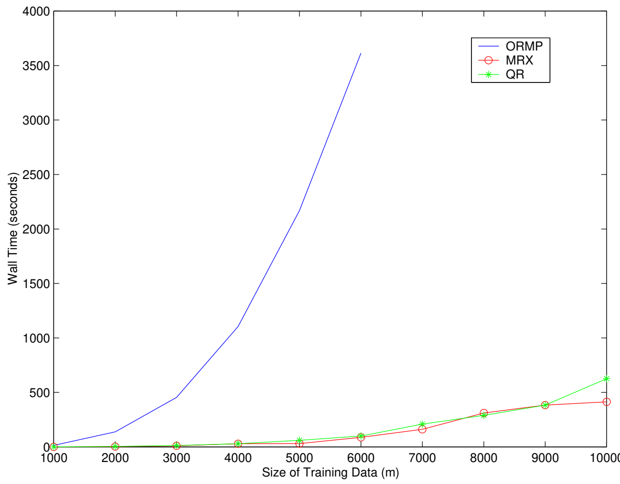

Figure 3. Comparison of the training time required by various approaches as a function of the size of the training dataset. The subset size (n) is chosen to be 10 % of m.

For the Abalone dataset, we averaged our results over 10 random splits of the mother data into 3000 and 1177 training and testing points, respectively. Fig. 2 and Table. 3 show the convergence trends of the residual error and the mean square testing error, respectively, as the subset size is increased. We observe that the residuals in this case may differ by as much as 13% across the methods considered. In contrast, the corresponding testing errors summarized in Table 3 do not differ significantly.

The final experiment demonstrates the scaling performance of the algorithms presented as the size of the training dataset increases. This study was conducted for the Census dataset (available from the DELVE repository) for the House-Price-8H prototask. Fig. 3 shows the computational time as a function of the size of the training dataset for various algorithms. Note that the subset size (n) is set at 10% of the training size. Further note that the timings shown also include the kernel computation time. As expected, the MRX and the incremental QR update-based factorization schemes scale impressively when compared to ORMP. In fact, our ORMP simulation seemed to scale worse than what was theoretically expected. We believe this was primarily because our implementation was rich in BLAS Level-1 operations. In contrast, for the MRX and the QR-based algorithm, we could employ Level-2 BLAS routines more extensively.

To summarize, we observe that across all the datasets considered, the QR-based algorithm appears to be quite competitive with ORMP in terms of the testing errors and reduction in the norm of the residual error. However, the QR-based algorithm and MRX are computationally much cheaper, and require minimal memory as compared to ORMP.

7. Concluding Remarks

In this paper, we presented iterative greedy algorithms for constructing sparse nonlinear regression models. An attractive feature of the algorithms presented here is the modest memory requirements and low computational cost. In particular, the computational cost only depends on the size of the training data subset. This has been demonstrated through numerical studies on some benchmark problems. The numerical studies also suggest that the generalization abilities of the sparse regression models constructed using the proposed algorithms are competitive with other state-of-the-art regression techniques.

Since different datasets have widely varying characteristics, it would be difficult to make general statements or predictions about the superiority of one algorithm over the rest. However, if computationally efficiency is a major concern, it appears that the present algorithms are more attractive than ORMP, particularly for massive regression problems.

It remains to be seen how greedy sparse models relate to those constructed using the SVM framework given that both approaches attempt a tradeoff between empirical error and model complexity; see for example, Girosi (1998). It is also of interest to note that the present approach may be applied to classification problems. Lastly, connections with Friedman's generalized greedy function approximation framework (Friedman, 1999) merit further investigation.

References

Adler, J., Rao, B. D & Kreutz-Delgado, K. (1996). Comparison of basis selection methods. *Proceedings of the 30th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, *1*, 252-257.

Chen, S. S., Donoho, D. L., and Saunders, M. A. (1998). Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, *20*, 33-61.

Chow, E., & Saad, Y. (1997). Approximate inverse techniques for block-partitioned matrices. *SIAM Journal of Scientific and Statistical Computing*, *18*, 1657-1675.

Couvreur, C., & Bresler, Y. (1999). On the optimality of the backward greedy algorithm for the subset selection problem. *SIAM Journal on Matrix Analysis and Its Applications*, *21 (3)*, 797-808.

Evgeniou, T., Pontil M., & Poggio T. (2000). Regularization networks and support vector machines. *Advances in Computational Mathematics*, *13*, 1-50.

Friedman, J.H. (1999). Greedy Function Approximation: A Gradient Boosting Machine. International Mathematical Society, *Reitz Lecture*, 1999. http://www-stat.stanford.edu/~jhf

Girosi, F. (1998). An equivalence between sparse approximation and support vector machines. *Neural Computation*, *10*, 1455-1480.

Grote, M. J. & Huckle, T. (1997). Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, *18(3)*, 838-853.

Mallat, S. & Zhang, Z. (1993). Matching pursuit in a time-frequency dictionary. *IEEE Transactions on Signal Processing*, *41*, 3397–3415.

Mangasarian, O. L., & Musicant, D. R. (2000). Robust linear and support vector regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22(9)*, 950-955.

Natarajan, B. K. (1995). Sparse approximate solutions to linear systems. *SIAM Journal of Computing*, *25(2)*, 227-234.

Natarajan, B. K. (1999). On learning functions from noise-free and noisy examples via Occam's razor. *SIAM Journal of Computing*, *29(3)*, 712-727.

Schaback, R., & Wendland, H. (2000). Adaptive greedy techniques for approximate solution of large RBF systems. *Numerical Algorithms*, *24(3)*, 239-254.

Schölkopf, B., Smola, A. J., Williamson, R., & Bartlett, P. (2000). New support vector algorithms. *Neural Computation*, *12(5)*, 1207-1245.

Smola, A. J., & Bartlett, P. L. (2001). Sparse greedy Gaussian process regression. *Advances in Neural Information Processing Systems*.

Smola, A. J., & Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. *International Conference on Machine Learning*.

Vapnik, V. (1995). *The nature of statistical learning*. New York: Springer Verlag.