ESANN'2006 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 26-28 April 2006, d-side publi., ISBN 2-930307-06-4.

Efficient Forward Regression with Marginal Likelihood

Ping Sun and Xin Yao
CERCIA, School of Computer Science
University of Birmingham, Edgbaston Park Road, Birmingham, B15 2TT, UK

Abstract. We propose an efficient forward regression algorithm based on greedy optimization of marginal likelihood. It can be understood as a forward selection procedure which adds a new basis vector at each step with the largest increment to the marginal likelihood. The computational cost of our algorithm is linear in the number n of training examples and quadratic in the number k of selected basis vectors, i.e. $\mathcal{O}(nk^2)$. Moreover, our approach is only required to store a small fraction of all columns of the full design matrix. We compare our algorithm with the well-known Relevance Vector Machines (RVM) which also optimizes marginal likelihood iteratively. The results show that our algorithm can achieve comparable prediction accuracy but with significantly better scaling performance in terms of both computational cost and memory requirements.

1 Introduction

This paper considers the regression problem with a linear model. It can be described briefly as follows. Given a training dataset $\mathcal{D} = \{(x_1, t_1), ..., (x_n, t_n)\}$ where $t_i \in \mathbb{R}$, $x_i \in \mathbb{R}^d$, n is the size of $\mathcal{D}$ and d is the dimension of an example, we aim to infer a function $f(x)$ which can predict unseen data. Usually, the function $f(\cdot)$ is assumed to have a fixed structure and to depend on a set of weight parameters w. In particular, we consider a function that is linear w.r.t w, i.e.

$$f(x, w) = \sum_{j=1}^{m} w_j h_j(x) = w^\top h(x), \tag{1}$$

where $h(x) = [h_1(x), ..., h_m(x)]^\top$ is a vector of m fixed nonlinear map functions of input x and $w = [w_1, ..., w_m]^\top$. If we set $h(x) = [\mathcal{K}(x_1, x), ..., \mathcal{K}(x_n, x)]^\top$, in this case $m = n$ and where $\mathcal{K}(x, x_i)$ is some symmetric kernel function, the concerned problem can be popularly termed as kernel regression [6].

The weight vector w can be found by minimizing the squared loss function, $\sum_{i=1}^{n} |t_i - f(x_i, w)|^2 = \|t - Hw\|^2$, where $t = [t_1, ..., t_n]$ and H is the so-called design matrix of size $n \times m$ with $H_{ij} = h_j(x_i)$. Here we refer to columns of H as *basis vectors*. It is easily seen that the optimal solution to the squared loss function can be reached at $w = (H^\top H)^{-1} H^\top t$. However, it is sometimes more useful to obtain a sparse estimate of w in practice, which corresponds to select a small fraction of available m basis vectors. In kernel-based methods, the sparse model (1) induced by a sparse estimate w can greatly improve the generalization performance [6].

Before addressing the issue of how to achieve the sparseness, we firstly discuss the choice of loss function used in the sparse model construction process.

The most usual one is the squared loss employed in [16, 7], which typically decreases as the model size increases. Hence a separate model selection criterion (such as the minimum descriptive length, MDL) or cross-validation step is required to terminate the construction procedure in order to avoid overfitting. A better and more natural idea is to use the loss function, estimating the model generalization capability directly in the training process. Such examples include generalized cross-validation (GCV) [8], leave-one-out cross validation (LOOCV) [5] and marginal likelihood (ML) loss functions [13, 14]. This paper will focus on the last one and the work can be extended to compare with other kinds of loss functions in the future.

A number of algorithms in the literature have been proposed to obtain sparse estimates. They can broadly be classified into sequential forward (or backward) greedy algorithms [1] and LASSO-style (least absolute shrinkage and selection operator) algorithms [12, 2]. The LOOCV-based algorithm mentioned above is typically in the category of forward algorithms; while GCV- [8] and ML-based algorithms [13] addressed before can be interpreted as LASSO-style approaches [11]. In general, LASSO-style algorithms are computationally more expensive than sequential forward greedy algorithms.

In this paper, we present a sequential forward algorithm applied to greedy optimization of the ML loss function. In contrast to Tipping's work [13, 15], the relevance vector machine (RVM) which was also optimizing the ML loss function to obtain the sparseness *but with LASSO-style*, the technique presented in this paper has more attractive computational and storage properties.

We now outline the contents of the paper. In Section 2, we describe our sequential forward algorithm in detail and analyze its time and space complexity. In Section 3, we clarify the relations to other similar work. Experimental results are reported in Section 4. Finally, Section 5 concludes the paper by presenting possible directions of future research.

The following notations have been used throughout this paper. Given one sequence of indices I, the notation $H(:, I)$ denotes the submatrix of the columns of H indexed by I, and similarly for $H(I, :)$. We let $Id_q \in \mathbb{R}^{q \times q}$ denote the identity matrix. The function '`diag`' outputs the diagonal elements of a square matrix and $|\cdot|$ denotes the determinant. Furthermore, the subscript k will be used to denote the number of selected basis vectors.

2 Forward Selection Algorithm

The ML loss function employed in our algorithm can be written as the negative logarithm of marginal likelihood [14], i.e.,

$$\begin{aligned} J &= \frac{1}{2}\left[t^\top(\sigma^2 Id_n + \alpha^{-1}HH^\top)^{-1}t + \log|\sigma^2 Id_n + \alpha^{-1}HH^\top|\right] \\ &= \frac{1}{2\sigma^2}t^\top Pt - \frac{\log\sigma^2}{2}\log|P| \end{aligned} \tag{2}$$

where $\sigma^2, \alpha > 0$ are two hyper-parameters and $P = Id_n - H\Sigma H^\top$ where $\Sigma = (H^\top H + \lambda Id_m)^{-1}$ with the so-called regularization parameter $\lambda = \sigma^2\alpha$.

Assuming that we have already collected $(k-1)$ basis vectors indexed by $I_{k-1} = (i_1, ..., i_{k-1})$, i.e., $H(:, I_{k-1})$, the k-th iteration then involves choosing a previously unselected column $H(:, i_k)$ from H such that J is minimized. Since the loss function after the k-th iteration is

$$\begin{aligned} J_k &= \frac{1}{2\sigma^2} t^\top P_k t - \frac{\log \sigma^2}{2} \log |P_k| \\ &= \frac{1}{2\sigma^2} \left(t^\top P_{k-1} t - \frac{A_{k-1}(i_k)^2}{\lambda + B_{k-1}(i_k)} \right) - \frac{\log \sigma^2}{2} \left(\log |P_{k-1}| + \log \frac{\lambda}{\lambda + B_{k-1}(i_k)} \right) \\ &= J_{k-1} - \Delta J_{k-1}(i_k), \end{aligned} \tag{3}$$

where

$$A_{k-1}(i_k) = H(:, i_k)^\top P_{k-1} t, \quad B_{k-1}(i_k) = H(:, i_k)^\top P_{k-1} H(:, i_k), \tag{4}$$

and

$$\Delta J_{k-1}(i_k) = \frac{1}{2\sigma^2} \cdot \frac{A_{k-1}(i_k)^2}{\lambda + B_{k-1}(i_k)} + \frac{\log \sigma^2}{2} \cdot \log \frac{\lambda}{\lambda + B_{k-1}(i_k)} \tag{5}$$

is the loss reduction, the index of the k-th basis vector to be included is selected as $i_k = \arg\max_{i \notin I_{k-1}} \{\Delta J_{k-1}(i)\}$. If we are given A_{k-1} and B_{k-1}, it is clear that the computation of ΔJ_{k-1} is very fast.

To proceed this forward selection process, we need to update $A_{k-1}(i)$, $B_{k-1}(i)$ for all $i \notin I_k$ from the $(k-1)$-th state to the k-th. To this end, it is also required to update the matrix P_{k-1}, which can be realized by modifying $M_{k-1} \in \mathbb{R}^{n \times (k-1)}$ and diagonal D_{k-1} in $P_{k-1} = Id_n - M_{k-1} D_{k-1} M_{k-1}^\top$. The steps involved at the k-th iteration are summarized below:

$$M_k = [M_{k-1} \;\; m_k], \quad m_k = H(:, i_k) - M_{k-1}(D_{k-1}(M_{k-1}^\top H(:, i_k))), \tag{6}$$

$$\texttt{diag}(D_k) = [\texttt{diag}(D_{k-1}) \;\; d_k^{-1}], \quad d_k = \lambda + H(:, i_k)^\top m_k, \tag{7}$$

$$e_k = t^\top m_k, \quad I_k = [I_{k-1} \;\; i_k], \tag{8}$$

$$A_k(i) = A_{k-1}(i) - C(i) e_k / d_k, \quad B_k(i) = B_{k-1}(i) - C(i)^2 / d_k, \quad i \notin I_k, \tag{9}$$

where

$$C(i) = H(:, i)^\top m_k, \quad i \notin I_k. \tag{10}$$

Obviously, the major computational cost incurred in our forward algorithm is the step of computing $C(i)$ for all $(m-k)$ available basis vectors and leads to a prohibitive $\mathcal{O}(nm)$ operation at each iteration. The overall complexity would be $\mathcal{O}(nmk)$ when the model construction process terminates at the k-th iteration. Furthermore, computing $C(i)$ still requires the computation and storage of the full design matrix H. For a large dataset, these could dramatically increase the training time and memory consumption.

Now we consider a special case where the design matrix H is set to a kernel matrix K generated by evaluating $\mathcal{K}(x_i, x_j)$ on the paired input vectors $\{(x_i, x_j), i, j = 1, ..., n\}$. If $\mathcal{K}(x_i, x_j)$ is a Gaussian, the step (10) of computing the product Kc with $c \in \mathbb{R}^n$ can be approximated quite efficiently and effectively. Firstly, the computation of Kc is actually the well-known Fast Gauss Transform

(FGT) [10] problem in the field of computer vision. Using FGT technique could improve the complexity from original $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ cost. Secondly, we can also accelerate computing Kc by finding a low-rank approximation Q of the kernel matrix K. Due to the space limit, we only investigate the second method in this paper. Standard algorithms [3] from numerical linear algebra can be exploited to compute an approximation of the form $Q = GG^\top$, where $G \in \mathbb{R}^{n\times p}$, and where the rank $p \ll n$ usually. The resulting computational complexity generally scales as $\mathcal{O}(np^2)$. Instead of computing Kc, we use the approximation $Qc = G(G^\top c)$ to replace it, which only incurs a $\mathcal{O}(np)$ operation instead of $\mathcal{O}(n^2)$. If we restrict p to be the same as the maximal allowed number $k_{\mathtt{max}}$ of selected basis vectors, the overall complexity of our algorithm keeps $\mathcal{O}(nk_{\mathtt{max}}^2)$ and the storage is $\mathcal{O}(nk_{\mathtt{max}})$. The property of linear scaling in n makes the presented work viable for large-scale problems.

3 Comparisons with RVM

The RVM technique was introduced in [13] and adopted a different mechanism to arrive at a sparse model. It can be reformulated as iteratively optimizing a variant of the ML-based loss function (2), that is,

$$J^{\mathtt{rvm}} = \frac{1}{2\sigma^2} t^\top P^{\mathtt{rvm}} t - \frac{\log \sigma^2}{2} \log |P^{\mathtt{rvm}}| \quad \text{with} \quad P^{\mathtt{rvm}} = Id_n - H\Sigma^{\mathtt{rvm}} H^\top, \tag{11}$$

where $\Sigma^{\mathtt{rvm}} = (H^\top H + \Lambda)^{-1}$ and Λ is a diagonal with elements $\mathtt{diag}(\Lambda) = [\lambda_1 \ \ldots \ \lambda_m]$. The loss function (11) uses multiple regularization parameters, which are ultimately responsible for the sparsity property of the RVM algorithm. This is because the optimum value for λ_i can be infinite, which is equivalent to removing the i-th basis vector, as the iterative optimization proceeds. However, the computation required for the RVM is very high and scales as $\mathcal{O}(n^3)$. A later improvement [15] was proposed to speed up the training of RVM. The idea is to optimize each λ_i in turn while the others are fixed and it will incur $\mathcal{O}(n^2 q)$ cost, where q is the number of updating iterations. Furthermore, their updating steps could lead to serious instabilities due to round-off error accumulations [4]. Finally, the computation and storage of a full kernel matrix is needed for both RVM algorithms.

4 Numerical Experiments

This section will compare the algorithm developed in this paper against other two RVM algorithms mentioned in Section 3 to verify the usefulness of our algorithm. All the algorithms were coded in Matlab 7.0 and run on a machine with PIV 2G and 512M memory. To evaluate generalization performance, we ultilize test *Mean Square Error* (MSE) given by $\frac{1}{n_{\mathtt{tst}}} \sum_{i=1}^{n_{\mathtt{tst}}} (t_i - f(x_i))^2$ where $n_{\mathtt{tst}}$ is the number of test examples and $f(x_i)$ is the predictive mean. For all experiments, we use the Gaussian kernel defined by $\mathcal{K}(x_i, x_j) = \exp\{\frac{|x_i - x_j|^2}{r}\}$ with the parameter $r > 0$. The parameter σ^2 is set to a fixed positive value

(e.g. 1). Other parameters including λ, r and $k_{\texttt{max}}$ are estimated via five-fold cross validation on a subset of 1000 examples randomly selected from original dataset. Note that 'Forward' is used to denote our algorithm, 'Old RVM' denotes original RVM [13] and 'New RVM' denotes improved RVM [15].

The first comparison study is performed on three real world datasets viz. Ozone, Boston and Abalone available from the *UCI machine learning repository*. Each of them is randomly partitioned into 20 training/test data splits. Table 1 reports the test set performances of the three methods on the three datasets. Across all datasets considered, the MSE results of our algorithm and the best of the two RVM methods are quite comparable, which is decided by the paired t-test with a p-value threshold of 0.01. During the experiment, we observe that New RVM method suffers from numerical instability since it produces very bad results for 3 out of 20 realizations on the Abalone dataset. The results given in Table 1 have deleted these bad entries (in italic). In contrast to our forward algorithm, RVM algorithms produce more sparse models than ours but it is still arguing whether the high sparsity is useful or not [9].

Table 1: The results of three algorithms on the three benchmark regression datasets. The mean squared error (MSE) with standard deviation over 20 realizations are reported. d denotes the number of input dimension. The number in parentheses reflects the number of selected basis vectors.

Dataset	d	n	$n_{\texttt{tst}}$	Forward	Old RVM	New RVM
Ozone	8	250	80	16.33±2.91 (100)	16.66 ± 2.79 (8)	16.39±2.79 (7)
Boston	12	481	25	7.76±3.99 (150)	7.75±3.96 (60)	7.78±3.60 (78)
Abalone	8	1000	3177	0.438±0.009 (100)	0.444±0.009 (25)	*0.489±0.029*(85)

We then demonstrate the scaling performance of the algorithms presented as the size of training data increases. This study was conducted for the California Housing dataset (available from *StatLib repository*) which has 20640 examples. Of those, 10640 instances are used to test and others are employed to construct 10 training datasets with the size ranging from 1000 to 10000. We set the parameter $k_{\texttt{max}}$ for our forward algorithm at the minimum between 10% of the training size and 500. Figure 1 shows the computational time and test NMSE (normalized MSE) of the three methods for varying training data sizes. As expected, our algorithm scales quite impressively with comparable prediction accuracy when compared to old and new RVMs. Note that the upper limits of old and new RVMs are just 2500 and 4500, respectively.

5 Conclusions

We propose an efficient forward kernel modeling algorithm based directly on optimizing marginal likelihood. This has been achieved by adopting a new forward selection scheme and efficient recursive updating steps. This work can be extended to other types of kernel-based methods such as Gaussian Process Regression (GPR) by changing the loss function. A detailed comparison study with other sparse algorithms will be reported in the future work.

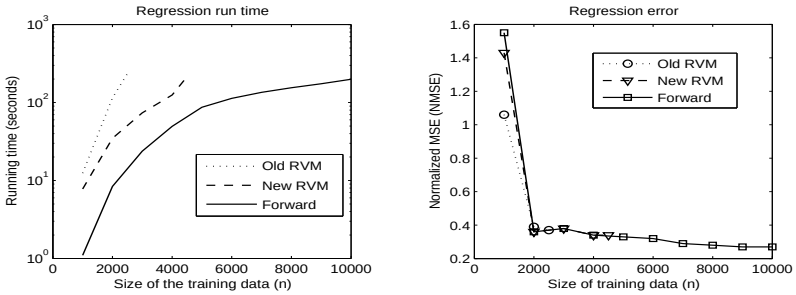

Fig. 1: The computational time and test NMSE for different approaches on the California Housing dataset.

Acknowledgments

The authors are grateful to Mike Tipping for his Matlab code of new RVM; to Pete Duell for reading a draft of this paper. This work is partially supported by ORS and the School of CS through PhD studentship to the first author.

References

[1] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning for radial basis function networks. *IEEE Trans. on NN*, 2(2):302–309, 1991.

[2] M. A. T. Figueiredo. Adaptive sparseness for supervised learning. *IEEE Trans. on PAMI*, 25(9):1150–1159, 2003.

[3] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *JMLR*, 2:243–264, 2002.

[4] G. H. Golub and C. V. Loan. *Matrix Computations*. Johns Hopkins Univ. Press, 1996.

[5] X Hong, P M Sharkey, and K Warwick. A robust nonlinear identification algorithm using PRESS statistic and forward regression. *IEEE Trans. on NN*, 14(2):454–458, 2003.

[6] K.-R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Trans. on NN*, 12(2):181–201, 2001.

[7] P. B. Nair, A. Choudhury, and A. J. Keane. Some greedy learning algorithms for sparse regression and classification with mercer kernels. *JMLR*, 3:781–801, 2002.

[8] Mark J. L. Orr. Local smoothing of RBF networks. In *ISANN*, Hsinchu, Taiwan, 1995.

[9] C E Rasmussen and J Quinonero-Candela. Healing the Relevance Vector Machine through Augmentation. In *ICML '05*, pages 689–696, 2005.

[10] V. C. Raykar, C. Yang, R. Duraiswami, and N. Gumerov. Fast Computation of Sums of Gaussians in High Dimensions. Technical report, UM CS Department, 2005.

[11] Volker Roth. The generalized LASSO. *IEEE Trans. on NN*, 15(1):16–27, 2004.

[12] R. Tibshirani. Regression shrinkage and selection via the LASSO. *J. Royal Statistical Soc. (B)*, 58:267–288, 1996.

[13] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *JMLR*, 1:211–244, 2001.

[14] M. E. Tipping. Bayesian inference: An introduction to principles and practice in machine learning. In *Advanced Lectures on Machine Learning*, pages 41–62, 2003.

[15] M. E. Tipping and A. Faul. Fast marginal likelihood maximisation for sparse bayesian models. In *AISTAT2003*, Key West, FL, 2003.

[16] P. Vincent and Y. Bengio. Kernel matching pursuit. *Mach. Lear.*, 48(1-3):165–187, 2002.