Efficient Hold-Out for Subset of Regressors

Tapio Pahikkala, Hanna Suominen, Jorma Boberg, and Tapio Salakoski

Turku Centre for Computer Science (TUCS)
University of Turku, Department of Information Technology
Joukahaisenkatu 3-5 B, FIN-20520 Turku, Finland
firstname.lastname@utu.fi

Abstract. Hold-out and cross-validation are among the most useful methods for model selection and performance assessment of machine learning algorithms. In this paper, we present a computationally efficient algorithm for calculating the hold-out performance for sparse regularized least-squares (RLS) in case the method is already trained with the whole training set. The computational complexity of performing the hold-out is $O(|H|^3 + |H|^2 n)$, where $|H|$ is the size of the hold-out set and n is the number of basis vectors. The algorithm can thus be used to calculate various types of cross-validation estimates effectively. For example, when m is the number of training examples, the complexities of N-fold and leave-one-out cross-validations are $O(m^3/N^2 + (m^2 n)/N)$ and $O(mn)$, respectively. Further, since sparse RLS can be trained in $O(mn^2)$ time for several regularization parameter values in parallel, the fast hold-out algorithm enables efficient selection of the optimal parameter value.

1 Introduction

In this paper, we consider the regularized least-squares (RLS) algorithm (see, e.g., [1]), a kernel-based learning method that is also known as the kernel ridge regression [2], least-squares support vector machine [3], and Gaussian process regression (see, e.g., [4]). RLS has been shown to have a state-of-the-art performance regression and classification and it has been applied in various practical tasks in which kernel based learning algorithms are needed (see e.g. [5]). It has also been modified for other problems such as ranking [6,7].

This popular machine learning method has, however, an inefficiency limitation in large-scale problems; the computational complexity of training an RLS learner together with a nonlinear kernel function is $O(m^3)$, where m is the number of training examples. This may be too tedious when the number of training examples is large.

To make the RLS algorithm more efficient, sparse versions have been considered. In these only a subset of training examples, often called the basis vectors, are used as regressors while the whole set is still used in the training process. This decreases the training complexity to $O(mn^2)$, where $n << m$ is the number of basis vectors (see, e.g., [1,8]). Here, we use the term sparse RLS when referring to RLS with the subset of regressors approach.

M. Kolehmainen et al. (Eds.): ICANNGA 2009, LNCS 5495, pp. 350–359, 2009.

In addition to deriving faster machine learning algorithms, it is essential to develop computationally efficient methods for the related performance estimation and model selection. In particular, fast cross-validation (CV) algorithms and their approximations have been proposed (see, e.g., [7,9,10,11,12]) for the RLS-based learning algorithms. In CV, a part of the training data is held out from training to be used for testing and this hold-out procedure is repeated several times. For more thorough discussion about the CV methods and their critical points, we refer to [13].

But how to perform CV efficiently with sparse RLS? Firstly, sparse RLS regression can also be considered as a standard RLS regression using a certain type of modified kernel function [14]. This allows the use of the efficient hold-out algorithms for standard RLS proposed by [10,11]. The computational complexity of these are $O(|H|^2 m)$, where $|H|$ is the size of the hold-out set. However, the presence of the coefficient m may make the algorithms too expensive in practice, especially if it is used to calculate CV estimates. For example, the computational complexity of leave-one-out CV (LOOCV) using this approach would be $O(m^2)$, which is more expensive than the training process of sparse RLS if $m > n^2$.

The second approach is to design CV algorithms especially for sparse RLS. Recently, [9] proposed this kind of LOOCV algorithm. Its computational complexity is $O(mn^2)$ which makes it much more practical than the LOOCV algorithm of standard RLS used together with the modified kernel function.

In this paper, we propose an even faster algorithm for computing hold-out estimates for sparse RLS. Its computational complexity is $O(|H|^3 + |H|^2 n)$. Consequently, our algorithm can be used to calculate various types of CV estimates efficiently. For example, when the sizes of the hold-out sets are sufficiently small, the computational complexity of N-fold CV is no larger than that of training sparse RLS. This is the case especially for LOOCV, whose computational complexity is only $O(mn)$. Further, our hold-out algorithm can be used to efficiently select the optimal regularization parameter value, because it can be combined with the fast method for training sparse RLS with several parameter values in parallel.

2 Sparse Regularized Least-Squares

We first formalize the methods of RLS and the subset of regressors method. We start by considering the hypothesis space $\mathcal{H}$. For this purpose we define so-called kernel functions. Let $\mathcal{X}$ denote the input space, which can be any set, and $\mathcal{F}$ denote an inner product space we call the feature space. For any mapping $\Phi : \mathcal{X} \to \mathcal{F}$, the inner product $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$ of the images of the data points $x, x' \in \mathcal{X}$ is called a kernel function.

Using k, we define for a set $X = \{x_1, \ldots, x_m\}$ of data points a symmetric kernel matrix $K \in \mathbb{R}^{m \times m}$, whose entries are given by $K_{i,j} = k(x_i, x_j)$ and $\mathbb{R}^{m \times m}$ denotes the set of real $m \times m$ -matrices. For simplicity, we assume that K is strictly positive definite, that is, $A^{\mathrm{T}} K A > 0$ for all $A \in \mathbb{R}^m, A \neq 0$. The strict positive definiteness of K can be ensured, for example, by adding ϵI to K, where $I \in \mathbb{R}^{m \times m}$ is the identity matrix and ϵ is a small positive real number.

We consider the RLS algorithm as a variational problem (for a more comprehensive introduction, see, e.g., [1])

$$h = \operatorname*{argmin}_{h \in \mathcal{H}} \left(\sum_{i=1}^{m} (h(x_i) - y_i)^2 + \lambda \|h\|_k^2 \right), \tag{1}$$

where y_i are the output labels corresponding to the training inputs x_i, $\|\cdot\|_k$ is the norm in the reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ determined by the kernel function k (see, e.g., [15]). The first and the second term of the right hand side of (1) are called the cost function and the regularizer, respectively, and $\lambda \in \mathbb{R}_+$ is a regularization parameter.

The solution of (1) has, by the representer theorem (see, e.g., [15]), the form

$$h(x) = \sum_{i=1}^{m} a_i k(x, x_i), \tag{2}$$

where coefficients $a_i \in \mathbb{R}$. Accordingly, we only need to solve a regularization problem with respect to a finite number of $a_i, 1 \leq i \leq m$. Let $A = (a_1, \ldots, a_m)^{\mathrm{T}} \in \mathbb{R}^m$ be a vector determining the minimizer of h.

To express (1) in a matrix form, we overload our notation and write $h(X) = KA \in \mathbb{R}^m$. This column vector contains the label predictions of the training data points obtained with the function h. Further, according to the properties of the RKHS determined by k, the regularizer can be written as $\lambda \|h\|_k^2 = \lambda A^{\mathrm{T}} K A$. Now, we can rewrite the algorithm (1) as

$$A = \operatorname*{argmin}_{A \in \mathbb{R}^m} \left((Y - KA)^{\mathrm{T}} (Y - KA) + \lambda A^{\mathrm{T}} K A \right).$$

Its solution

$$A = (KK + \lambda K)^{-1} KY = (K + \lambda I)^{-1} Y \tag{3}$$

is found by first differentiating $(Y - KA)^{\mathrm{T}}(Y - KA) + \lambda A^{\mathrm{T}} K A$ with respect to A, setting the derivative to be zero, and solving it with respect to A.

The computation complexity of calculating the coefficient vector A from (3) is dominated by the inversion of a $m \times m$ -matrix, and hence it can be performed in $O(m^3)$ time. This may be too tedious when the number of training examples is large. However, several authors have considered sparse versions of RLS, where only a part of the training examples, called basis vectors, have a nonzero coefficient in (2). This means that when the training is complete, only the basis vectors are needed when predicting the outputs of the new data points, which makes the prediction more efficient than it is with the standard kernel RLS regression. Another advantage of sparse RLS is that its training complexity is only $O(mn^2)$, where n is the number of basis vectors. Further, as we will show below, there are efficient CV and regularization parameter selection algorithms for sparse RLS that are analogous to the ones for standard RLS.

In this paper, we do not pay attention to the approach that is used to select the set of basis vectors. The only detail we point out is related to the computation of the hold-out or CV performance, since this is our main topic. Namely, the selection of the basis vectors should not change if a part of the training examples is held out and sparse RLS is trained with the rest of the examples. One suitable selection method is, for example, a random sub-sampling when the hold-out sets are also selected randomly. In fact, it was found in [1] that simply selecting the basis vectors randomly has no worse learning performance than more sophisticated methods.

Before continuing to the definition of sparse RLS, we introduce some notation. Let $\mathcal{M}_{\Xi\times\Psi}$ denote the set of matrices whose rows and columns are indexed by the index sets Ξ and Ψ, respectively. Below, with any matrix $M \in \mathcal{M}_{\Xi\times\Psi}$ and index set $\Upsilon \subseteq \Xi$, we use the subscript Υ so that a matrix $M_\Upsilon \in \mathcal{M}_{\Upsilon\times\Psi}$ contains only the rows that are indexed by Υ. For $M \in \mathcal{M}_{\Xi\times\Psi}$, we also use $M_{\Upsilon\Omega} \in \mathcal{M}_{\Upsilon\times\Omega}$ to denote a matrix that contains only the rows and the columns that are indexed by any index sets $\Upsilon \subseteq \Xi$ and $\Omega \subseteq \Psi$, respectively.

We now follow [8,1,14] and define the sparse RLS algorithm using the above defined notation. Let $F = \{1, \ldots, m\}$. Instead of allowing functions like in (2), we only allow

$$h(x) = \sum_{i\in B} a_i k(x, x_i),$$

where the set indexing the n basis vectors $B \subset F$ is selected in advance. In this case, the coefficient vector $A \in \mathbb{R}^n$ is a vector whose entries are indexed by B. The label predictions for the training data points can be obtained from

$$h(X) = (K_B)^{\mathrm{T}} A \tag{4}$$

and the regularizer can be rewritten as $\lambda A^{\mathrm{T}} K_{BB} A$. Therefore, the coefficient vector A is the minimizer of

$$(Y - (K_B)^{\mathrm{T}} A)^{\mathrm{T}} (Y - (K_B)^{\mathrm{T}} A) + \lambda A^{\mathrm{T}} K_{BB} A. \tag{5}$$

The minimizer of (5) is found by setting its derivative with respect to A to zero. It is

$$A = P^{-1} K_B Y, \tag{6}$$

where

$$P = K_B (K_B)^{\mathrm{T}} + \lambda K_{BB} \in \mathcal{M}_{B\times B}. \tag{7}$$

The matrices K_{BB} and $K_B(K_B)^{\mathrm{T}} = (KK)_{BB}$ are principal sub-matrices of the positive definite matrices K and KK, respectively, and hence the matrix P is also positive definite and invertible (see e.g. [16, p. 397]). In contrast to (3), the matrix inversion involved in (6) can be performed in $O(n^3)$ time. Since $n << m$, the overall computational complexity of (6) is dominated by the complexity of calculating $K_B(K_B)^{\mathrm{T}}$ which is $O(mn^2)$.

We now reformulate sparse RLS so that its coefficient matrix can be efficiently calculated for different values of the regularization parameter. Note that this reformulation is already known in the machine learning community. Let

$$K_{BB} = CC^{\mathrm{T}} \tag{8}$$

be the Cholesky factorization of K_{BB}, where $C \in \mathcal{M}_{B\times B}$ is a lower triangular matrix with strictly positive diagonal entries. Moreover, let

$$C^{-1}K_B(K_B)^{\mathrm{T}}(C^{-1})^{\mathrm{T}} = V\Lambda V^{\mathrm{T}} \tag{9}$$

be the eigen decomposition of $C^{-1}K_B(K_B)^{\mathrm{T}}(C^{-1})^{\mathrm{T}}$, where $V \in \mathcal{M}_{B\times B}$ is the matrix containing the eigenvectors, and $\Lambda \in \mathcal{M}_{B\times B}$ is a diagonal matrix containing the eigenvalues of the decomposition. Further, let

$$\widetilde{\Lambda} = (\Lambda + \lambda I)^{-1}$$

and

$$Q = (C^{-1})^{\mathrm{T}}V \in \mathcal{M}_{B\times B}. \tag{10}$$

Then, the matrix P^{-1} can be expressed as

$$\begin{aligned} P^{-1} &= (K_B(K_B)^{\mathrm{T}} + \lambda K_{BB})^{-1} \\ &= (K_B(K_B)^{\mathrm{T}} + \lambda CC^{\mathrm{T}})^{-1} \\ &= (C^{-1})^{\mathrm{T}}(C^{-1}K_B(K_B)^{\mathrm{T}}(C^{-1})^{\mathrm{T}} + \lambda I)^{-1}C^{-1} \\ &= (C^{-1})^{\mathrm{T}}(V\Lambda V^{\mathrm{T}} + \lambda I)^{-1}C^{-1} \\ &= (C^{-1})^{\mathrm{T}}V\widetilde{\Lambda}V^{\mathrm{T}}C^{-1} \\ &= Q\widetilde{\Lambda}Q^{\mathrm{T}}. \end{aligned} \tag{11}$$

The computational complexities of calculating (8), (9) and (10) are $O(n^3)$.

Now, let us first calculate $Q^{\mathrm{T}}K_BY$ (in $O(nm)$ time) and store it in memory. Then, the solution (6) can be computed for different values of the regularization parameter from

$$A = Q\widetilde{\Lambda}Q^{\mathrm{T}}K_BY,$$

with a complexity $O(n^2)$. This is because the multiplication of the shifted and inverted eigenvalues $\widetilde{\Lambda}$ with $Q^{\mathrm{T}}K_BY$ can be performed in $O(n)$ time and the multiplication of the resulting matrix from left by Q can be performed in $O(n^2)$ time.

3 Fast Computation of Hold-Out Error Estimates for Sparse RLS

We note (see, e.g., [14]) that the sparse approach can also be considered as performing a standard RLS regression using the following type of modified kernel function

$$\tilde{k}(x, x') = k(x, X)(K_{BB})^{-1}k(X, x), \tag{12}$$

Therefore, a straightforward way to construct hold-out estimates for sparse RLS would be to use the hold-out algorithms proposed by [10,11] for the standard RLS regression using the modified kernel function (12). The computational complexity of this approach is $O(|H|^2 m)$, where m is the number of training examples and $|H|$ is the size of the hold-out set. The presence of the coefficient m may make it computationally too expensive in practice, especially if it is used to calculate CV estimates. Further, it can be shown that if a data point in the hold-out set is a basis vector, its effect is not completely removed from the training process, because the kernel (12) depends on it.

We now introduce a faster algorithm for calculating a hold-out performance estimates, whose computational complexity is $O(|H|^2(|H|+n))$, where $n << m$ is the number of basis vectors. Consequently, our algorithm can be used to calculate various types of CV estimates efficiently. For example, when the sizes of the hold-out sets are sufficiently small, the computational complexity of N-fold CV is no larger than that of training sparse RLS. This is the case especially for LOOCV.

As previously, $F = \{1, \ldots, m\}$ and $B \subset F$ are the index set for the whole training data set and the set indexing the basis vectors, respectively. Let $H \subset F$ denote the set of indices of the hold-out data points, and let $\overline{H} = F \setminus H$, $E = H \cap B$, and $L = \overline{H} \cap B$. Further, let $h_{\overline{H}}$ be the function obtained by training the sparse RLS algorithm without using the training examples indexed by H. Then, $h_{\overline{H}}(X_H)$ consists of the output values for the hold-out data points X_H that are predicted by $h_{\overline{H}}$. According to (6), the coefficient vector corresponding to $h_{\overline{H}}$ is

$$G^{-1} K_{L\overline{H}} Y_{\overline{H}},$$

where

$$G = K_{L\overline{H}} K_{\overline{H}L} + \lambda K_{LL}.$$

The entries of this coefficient vector are indexed by L. Therefore, according to (4), the output values corresponding to the hold-out set H can be obtained from

$$h_{\overline{H}}(X_H) = K_{HL} G^{-1} K_{L\overline{H}} Y_{\overline{H}}. \tag{13}$$

This is, of course, computationally too expensive to be used in CV, but fortunately, it is possible to calculate the outputs more efficiently when we have trained in advance a sparse RLS learner with the whole data set.

Proposition 1. *Suppose that we have trained sparse RLS by calculating (8), (9), and (10), and we have the following matrices stored in memory:*

$$K_B \in \mathcal{M}_{B \times F} \tag{14}$$
$$\Lambda \in \mathcal{M}_{B \times B} \tag{15}$$
$$Q \in \mathcal{M}_{B \times B} \tag{16}$$
$$(K_B)^T Q \in \mathcal{M}_{F \times B} \tag{17}$$
$$K_B Y \in \mathcal{M}_{B \times 1} \tag{18}$$
$$Q^T K_B Y \in \mathcal{M}_{B \times 1}. \tag{19}$$

Then, the hold-out predictions for a set H can be calculated from

$$h_{\overline{H}}(X_H) = -(J - I)^{-1}R, \tag{20}$$

where

$$J = U\widetilde{\Lambda}U^T - U\widetilde{\Lambda}(Q_E)^T(Q_E\widetilde{\Lambda}(Q_E)^T)^{-1}Q_E\widetilde{\Lambda}U^T, \tag{21}$$

$$R = U\widetilde{\Lambda}Z - U\widetilde{\Lambda}(Q_E)^T(Q_E\widetilde{\Lambda}(Q_E)^T)^{-1}Q_E\widetilde{\Lambda}Z, \tag{22}$$

$$U = ((K_B)^TQ)_H - K_{HE}Q_E, \text{ and} \tag{23}$$

$$Z = Q^TK_BY - (Q_E)^T(K_BY)_E - (Q^TK_B)_{BH}Y_H + (Q_E)^TK_{EH}Y_H. \tag{24}$$

The computational complexity of this calculation is $O(|H|^2(|H| + n))$.

Proof. We start by showing the tenability of (20) and continue by considering the computational complexities. Recall from (13) that the output matrix for the hold-out set can be obtained from

$$h_{\overline{H}}(X_H) = K_{HL}G^{-1}K_{L\overline{H}}Y_{\overline{H}}, \tag{25}$$

where

$$\begin{aligned} G &= K_{L\overline{H}}K_{\overline{H}L} + \lambda K_{LL} \\ &= K_L(K_L)^{\mathrm{T}} - K_{LH}K_{HL} + \lambda K_{LL} \\ &= P_{LL} - K_{LH}K_{HL} \end{aligned}$$

and P is defined in (7). Now, due to the positive definiteness of K, both G and P_{LL} are always invertible (see e.g. [16]). Let

$$W = (P_{LL})^{-1}.$$

Using the block inverse formula (see, e.g., [16, p. 18–19]), we get

$$W = (P^{-1})_{LL} - (P^{-1})_{LE}((P^{-1})_{EE})^{-1}(P^{-1})_{EL}. \tag{26}$$

Now, we observe that

$$G^{-1} = (W^{-1} - K_{LH}K_{HL})^{-1},$$

and using the Sherman-Morrison-Woodbury formula (see, e.g., [16, p. 18–19]), we obtain

$$G^{-1} = W - WK_{LH}(-I + K_{HL}WK_{LH})^{-1}K_{HL}W. \tag{27}$$

The invertibility of the matrix $-I + K_{HL}WK_{LH}$ can be shown by considering the matrix

$$\begin{bmatrix} I & -K_{HL} \\ -K_{LH} & W^{-1} \end{bmatrix}. \tag{28}$$

Since $G = W^{-1} - K_{LH}K_{HL}$, and I are invertible, the invertibility of the matrix (28) follows from the Schur's determinantal formula (see, e.g., [16, p. 21]).

Therefore, also the matrix $I - K_{HL}WK_{LH}$ is invertible, again due to the Schur's determinantal formula.

We continue by observing that

$$\begin{aligned} U &= ((K_B)^{\mathrm{T}}Q)_H - K_{HE}Q_E \\ &= K_{HL}Q_L \text{ and} \qquad (29) \\ Z &= Q^{\mathrm{T}}K_BY - (Q_E)^{\mathrm{T}}(K_BY)_E - (Q^{\mathrm{T}}K_B)_{BH}Y_H + (Q_E)^{\mathrm{T}}K_{EH}Y_H \\ &= (Q_L)^{\mathrm{T}}K_LY - (Q_L)^{\mathrm{T}}K_{LH}Y_H \\ &= (Q_L)^{\mathrm{T}}K_{L\overline{H}}Y_{\overline{H}}. \qquad (30) \end{aligned}$$

According to (26), (11), and (29), we get

$$\begin{aligned} K_{HL}WK_{LH} &= K_{HL}(P^{-1})_{LL}K_{LH} - K_{HL}(P^{-1})_{LE}((P^{-1})_{EE})^{-1}(P^{-1})_{EL}K_{LH} \\ &= K_{HL}(Q\widetilde{\Lambda}Q^{\mathrm{T}})_{LL}K_{LH} \\ &\quad -K_{HL}(Q\widetilde{\Lambda}Q^{\mathrm{T}})_{LE}((Q\widetilde{\Lambda}Q^{\mathrm{T}})_{EE})^{-1}(Q\widetilde{\Lambda}Q^{\mathrm{T}})_{EL}K_{LH} \\ &= U\widetilde{\Lambda}U^{\mathrm{T}} - U\widetilde{\Lambda}(Q_E)^{\mathrm{T}}(Q_E\widetilde{\Lambda}(Q_E)^{\mathrm{T}})^{-1}Q_E\widetilde{\Lambda}U^{\mathrm{T}} \\ &= J. \end{aligned}$$

Analogously, according to (26), (11), and (30), we get $K_{HL}WK_{L\overline{H}}Y_{\overline{H}} = R$. Finally, by substituting (27) into (25), we get

$$\begin{aligned} h_{\overline{H}}(X_H) &= K_{HL}(W - WK_{LH}(J-I)^{-1}K_{HL}W)K_{L\overline{H}}Y_{\overline{H}} \\ &= (I - J(J-I)^{-1})R \\ &= ((J-I)(J-I)^{-1} - J(J-I)^{-1})R \\ &= -(J-I)^{-1}R. \end{aligned}$$

We now consider the computational complexity of using (20). The matrix U can be calculated from (14), (16), and (17) using (23) in $O(|H|^2n)$ time. Moreover, the matrix Z can be calculated from (14), (16), (17), (18), and (19) using (24) in $O(|H|n)$ time. The computational complexity of calculating J and R using (21) and (22) is $O(|H|^2(|H|+n))$. This is because multiplication of an $|H| \times n$ -matrix with a diagonal matrix $\widetilde{\Lambda}$ can be computed in $O(|H|n)$ time, the matrix inversion involved in the calculations needs $O(|H|^3)$ time, and all the other matrix products need at most in $O(|H|^2n)$ time when performed in the optimal order. Finally, we substitute these matrices in (20) from which the solution is obtained by inverting a matrix in $O(|H|^3)$ time. □

The calculation of the matrices (14) – (19) needs $O(mn^2)$ time, and hence the computational complexity of training sparse RLS as in Proposition 1 is the same as that of training sparse RLS in the ordinary way. When we have trained sparse RLS as in Proposition 1, we can use the efficient hold-out method for calculating various types of CV estimates. We can, for example, to perform N-fold CV by partitioning the training set into N approximately equally sized folds and average the results of the individual hold-out estimates. The number of training examples

in each fold is then $|H| \approx m/N$. According to Proposition 1, the computational complexity of each hold-out calculation is $O(|H|^2(|H|+n))$, and hence the overall computational complexity of N-fold CV is

$$O\left(m|H|^2 + mn|H|\right) = O\left(\frac{m^3}{N^2} + \frac{m^2 n}{N}\right).$$

We observe that when $|H| \approx n$, the computational complexity of CV is equal to that of the sparse RLS training. If we consider this as the largest tolerable computational complexity, the proposed hold-out method is too expensive for larger hold-out sets.

On the other hand, the method is less complex for smaller hold-out sets, like $O(mn)$ for the extreme case of LOOCV. Therefore, it can be used, for example, to select the value of the regularization parameter λ efficiently from a set of candidate values. The computational complexity of the regularization parameter selection is equal to the complexity of CV times the size of the set of candidate values, since the initialization phase of Proposition 1 does not have to be repeated for different candidate values.

4 Conclusion

Hold-out and CV are among the most important methods for model selection and performance evaluation of machine learning algorithms, and therefore their computationally efficient implementations are sought after. In this paper, we presented a computationally efficient algorithm for calculating hold-out performance estimates for sparse RLS when it has been trained in advance with the whole data set.

The computational complexity of training sparse RLS is $O(mn^2)$, where m is the size of the training set and n is the number of basis vectors. We showed that the hold-out estimates for trained sparse RLS can be computed in $O(|H|^2(|H|+n))$ time, where $|H|$ is the size of the hold-out set. Consequently, the algorithm can be used to calculate various types of CV estimates effectively. For example, the complexities of N-fold CV and LOOCV for m training examples are $O(m^3/N^2 + (m^2 n)/N)$ and $O(mn)$, respectively.

Sparse RLS can be trained in $O(mn^2)$ time for several regularization parameter values in parallel, and this property can also be combined with the fast hold-out calculation. Therefore, cross-validation can be used to efficiently select the optimal value of the regularization parameter.

Acknowledgments

This work has been supported by Tekes, the Finnish Funding Agency for Technology and Innovation (grant 40020/07) and by Academy of Finland (grant 128061).

References

1. Rifkin, R.: Everything Old Is New Again: A Fresh Look at Historical Approaches in Machine Learning. Ph.D thesis, Massachusetts Institute of Technology (2002)
2. Saunders, C., Gammerman, A., Vovk, V.: Ridge regression learning algorithm in dual variables. In: Proceedings of the Fifteenth International Conference on Machine Learning, pp. 515–521. Morgan Kaufmann Publishers Inc., San Francisco (1998)
3. Suykens, J.A.K., Vandewalle, J.: Least squares support vector machine classifiers. Neural Processing Letters 9(3), 293–300 (1999)
4. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press, Cambridge (2005)
5. Pahikkala, T., Pyysalo, S., Boberg, J., Järvinen, J., Salakoski, T.: Matrix representations, linear transformations, and kernels for disambiguation in natural language. Machine Learning 74(2), 133–158 (2009)
6. Pahikkala, T., Tsivtsivadze, E., Airola, A., Boberg, J., Salakoski, T.: Learning to rank with pairwise regularized least-squares. In: Joachims, T., Li, H., Liu, T.Y., Zhai, C. (eds.) SIGIR 2007 Workshop on Learning to Rank for Information Retrieval, pp. 27–33 (2007)
7. Pahikkala, T., Tsivtsivadze, E., Airola, A., Järvinen, J., Boberg, J.: An efficient algorithm for learning to rank from preference graphs. Machine Learning 75(1), 129–165 (2009)
8. Smola, A.J., Schölkopf, B.: Sparse greedy matrix approximation for machine learning. In: Langley, P. (ed.) Proceedings of the Seventeenth International Conference on Machine Learning, pp. 911–918. Morgan Kaufmann, San Francisco (2000)
9. Cawley, G.C., Talbot, N.L.C.: Fast exact leave-one-out cross-validation of sparse least-squares support vector machines. Neural Networks 17(10), 1467–1475 (2004)
10. Pahikkala, T., Boberg, J., Salakoski, T.: Fast n-fold cross-validation for regularized least-squares. In: Honkela, T., Raiko, T., Kortela, J., Valpola, H. (eds.) Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006), Espoo, Finland, Otamedia, pp. 83–90 (2006)
11. An, S., Liu, W., Venkatesh, S.: Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression. Pattern Recognition 40(8), 2154–2162 (2007)
12. Rifkin, R., Lippert, R.: Notes on regularized least squares. Technical Report MIT-CSAIL-TR-2007-025, Massachusetts Institute of Technology (2007)
13. Suominen, H., Pahikkala, T., Salakoski, T.: Critical points in assessing learning performance via cross-validation. In: Honkela, T., Pöllä, M., Paukkeri, M.S., Simula, O. (eds.) Proceedings of the 2nd International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR 2008), Helsinki University of Technology, pp. 9–22 (2008)
14. Quiñonero-Candela, J., Rasmussen, C.E.: A unifying view of sparse approximate gaussian process regression. Journal of Machine Learning Research 6, 1939–1959 (2005)
15. Schölkopf, B., Herbrich, R., Smola, A.J.: A generalized representer theorem. In: Helmbold, D., Williamson, R. (eds.) COLT 2001 and EuroCOLT 2001. LNCS, vol. 2111, pp. 416–426. Springer, Heidelberg (2001)
16. Horn, R., Johnson, C.R.: Matrix Analysis. Cambridge University Press, Cambridge (1985)