Recursive Finite Newton Algorithm for Support Vector Regression in the Primal

Liefeng Bo
blf0218@163.com
Ling Wang
wliiip@163.com
Licheng Jiao
lchjiao@mail.xidian.edu.cn
Institute of Intelligent Information Processing, Xidian University, Xi'an 710071, China

Some algorithms in the primal have been recently proposed for training support vector machines. This letter follows those studies and develops a recursive finite Newton algorithm (IHLF-SVR-RFN) for training nonlinear support vector regression. The insensitive Huber loss function and the computation of the Newton step are discussed in detail. Comparisons with LIBSVM 2.82 show that the proposed algorithm gives promising results.

1 Introduction

Support vector machines (SVMs) (Burges, 1998; Vapnik, 1998; Smola & Schölkopf, 2004) are powerful tools for classification and regression. In the past few years, fast algorithms for SVMs have been an active research direction. Traditionally, SVMs are trained by using decomposition techniques such as SVM$^{\text{light}}$ (Joachims, 1999) and sequential minimal optimization (Platt, 1999; Shevade, Keerthi, Bhattacharyya, & Murthy, 2000; Keerthi, Shevade, Bhattacharyya, & Murthy, 2001), which solve the dual problem by optimizing a small subset of the variables each iteration. Recently, some algorithms in the primal have been presented for training SVMs. (Mangasarian, 2002; Fung & Mangasarian, 2003) proposed the finite Newton algorithm and showed that it is rather powerful for linear SVMs. Keerthi and DeCoste (2005) introduced some appropriate techniques to speed up the finite Newton algorithm and named the resulting algorithm the modified finite Newton algorithm. Chapelle (2006) proposed the recursive finite Newton algorithm and showed it to be as efficient as the dual domain method for nonlinear support vector classification.

This letter follows these studies and develops a recursive finite Newton algorithm (IHLF-SVR-RFN) for nonlinear SVR. One of its main contributions is to introduce an insensitive Huber loss function that includes several popular loss functions as its special cases. Comparisons with LIBSVM 2.82 (Fan, Chen, & Lin, 2005) suggest that IHLF-SVR-RFN is as efficient as the dual domain method for nonlinear SVR.

The letter is organized as follows. In section 2, SVR in the primal is introduced. The insensitive Huber loss function and its corresponding optimization problem are proposed in section 3. The recursive finite Newton algorithm is discussed in section 4. Comparisons with LIBSVM 2.82 are reported in section 5. Section 6 presents the conclusions.

2 Support Vector Regression in the Primal

Consider a regression problem with training samples $\{\mathbf{x}_i, y_i\}_{i=1}^n$ where $\mathbf{x}_i$ is the input sample and y_i is the corresponding target. To obtain a linear predictor, SVR solves the following optimization problem:

$$\begin{aligned}
&\min_{\mathbf{w},b}\left(\frac{\|\mathbf{w}\|^2}{2}+C\sum_{i=1}^{n}\left(\xi_i^p+\bar{\xi}_i^p\right)\right)\\
&s.t.\ \mathbf{w}\cdot\mathbf{x}_i+b-y_i\le\varepsilon+\xi_i\\
&y_i\ -\mathbf{w}\cdot\mathbf{x}_i+b\le\varepsilon+\bar{\xi}_i\\
&\xi_i,\bar{\xi}_i\ge 0,\ i=1,2,\cdots,n.
\end{aligned}\tag{2.1}$$

Eliminating the slack variables $\left\{\xi_i,\bar{\xi}_i\right\}_{i=1}^n$ and dividing equation 2.1 by the factor C, we get the unconstrained optimization problem,

$$\min_{\mathbf{w},b}\left(L_\varepsilon(\mathbf{w},b)=\sum_{i=1}^{n}l_\varepsilon(\mathbf{w}\cdot\mathbf{x}_i+b-y_i)+\lambda\|\mathbf{w}\|^2\right),\tag{2.2}$$

where $\lambda=\frac{1}{2C}$ and $l_\varepsilon(r)=\max(|r|-\varepsilon,0)^p$. The most popular selections for p are 1 and 2. For convenience of expression, the loss function with $p=1$ is referred to as insensitive linear loss function (ILLF) and that with $p=2$ insensitive quadratic loss function (IQLF).

Nonlinear SVR can be obtained by using a kernel function and an associated reproducing kernel Hilbert space H. The resulting optimization is

$$\min_{f}\left(L_\varepsilon(f)=\sum_{i=1}^{n}l_\varepsilon(f(\mathbf{x}_i)-y_i)+\lambda\|f\|_H^2\right),\tag{2.3}$$

where we have dropped b for simplicity. Our experience shows that the generalization performance of SVR is not affected by this drop. According to the representer theory (Kimeldorf & Wahba, 1970), the optimal function for equation 2.3 can be expressed as a linear combination of the kernel functions centered in the training samples,

$$f(\mathbf{x}) = \sum_{i=1}^{n} \beta_i k(\mathbf{x}, \mathbf{x}_i). \tag{2.4}$$

Substituting equation 2.4 into 2.3, we have

$$\min_{\boldsymbol{\beta}} \left(L_\varepsilon(\boldsymbol{\beta}) = \sum_{i=1}^{n} l_\varepsilon \left(\sum_{j=1}^{n} \beta_i k\left(\mathbf{x}_i, \mathbf{x}_j\right) - y_i \right) + \lambda \sum_{i=1}^{n} \beta_i \beta_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) \right). \tag{2.5}$$

Introducing the kernel matrix $\mathbf{K}$ with $\mathbf{K}_{ij} = k\left(\mathbf{x}_i, \mathbf{x}_j\right)$ and $\mathbf{K}_i$ the ith row of $\mathbf{K}$, equation 2.5 can be rewritten as

$$\min_{\boldsymbol{\beta}} \left(L_\varepsilon(\boldsymbol{\beta}) = \sum_{i=1}^{n} l_\varepsilon(\mathbf{K}_i \boldsymbol{\beta} - y_i) + \lambda \boldsymbol{\beta}^T \mathbf{K} \boldsymbol{\beta} \right). \tag{2.6}$$

As long as $l_\varepsilon(\cdot)$ is differentiable, we can optimize equation 2.6 by a gradient descent algorithm.

3 Finite Newton Algorithm for Insensitive Huber Loss Function

The finite Newton algorithm is a Newton algorithm that can be proved to converge in a finite number of steps, and it has been demonstrated to be very efficient for support vector classification (Mangasarian, 2002; Keerthi & DeCoste, 2005; Chapelle, 2006). Here, we focus on the regression case. The finite Newton algorithm is straightforward for the insensitive quadratic loss function by defining the generalized Hessian matrix (Clarke, 1983; Hiriart-Urruty, Strodiot, & Nguyen, 1984; Keerthi & DeCoste, 2005); however, it is not applicable to the insensitive linear loss function since it is not differentiable. Inspired by the Huber loss function (Huber, 1981), we propose an insensitive Huber loss function (IHLF),

$$l_{\varepsilon,\Delta}(z) = \begin{cases} 0 & \textit{if} \quad |z| \le \varepsilon \\ (|z| - \varepsilon)^2 & \textit{if} \quad \varepsilon < |z| < \Delta, \\ (\Delta - \varepsilon)(2|z| - \Delta - \varepsilon) & \textit{if} \quad |z| \ge \Delta \end{cases} \tag{3.1}$$

whose shape is shown in Figure 1.

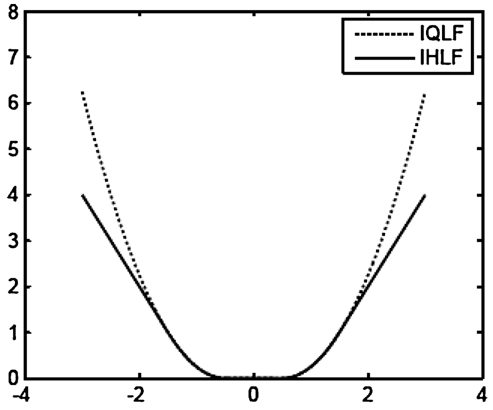

Figure 1: Insensitive Huber loss function with $\varepsilon = 0.5$ and $\Delta = 1.5$ and insensitive quadratic loss function with $\varepsilon = 0.5$.

We emphasize that Δ is strictly greater than ε, ensuring that IHLF is differentiable. Its first-order derivative can be written as

$$\frac{\partial l_{\varepsilon,\Delta}(z)}{\partial z} = \begin{cases} 0 & \textit{if} \quad |z| \le \varepsilon \\ 2\textit{sign}(z)(|z| - \varepsilon) & \textit{if} \quad \varepsilon < |z| < \Delta, \\ 2\textit{sign}(z)(\Delta - \varepsilon) & \textit{if} \quad |z| \ge \Delta \end{cases} \tag{3.2}$$

where $\textit{sign}(z)$ is 1 if $z \ge 0$; otherwise, $\textit{sign}(z)$ is -1.

The properties of IHLF are controlled by two parameters: ε and Δ. At certain ε and Δ values, we can obtain some familiar loss functions: (1) for $\varepsilon = 0$ and an appropriate Δ, IHLF becomes the Huber loss function (HLF); (2) for $\varepsilon = 0$ and $\Delta = \infty$, IHLF becomes the quadratic (gaussian) loss function (QLF); (3) for $\varepsilon = 0$ and $\Delta \to \varepsilon$, IHLF approaches the linear (Laplace) loss function (LLF); (4) for $0 < \varepsilon < \infty$ and $\Delta = \infty$, IHLF becomes the insensitive quadratic loss function (IQLF); and (5) for $0 < \varepsilon < \infty$ and $\Delta \to \varepsilon$, IHLF approaches the insensitive linear loss function (ILLF). The profiles of the loss functions derived from equation 3.1 are shown in Figure 2.

Introducing IHLF into the optimization problem, equation 2.6, we have the following nonlinear SVR problem:

$$\min_{\boldsymbol{\beta}} \left(L_{\varepsilon,\Delta}(\boldsymbol{\beta}) = \sum_{i=1}^{n} l_{\varepsilon,\Delta}(\mathbf{K}_i\boldsymbol{\beta} - y_i) + \lambda\boldsymbol{\beta}^T\mathbf{K}\boldsymbol{\beta} \right). \tag{3.3}$$

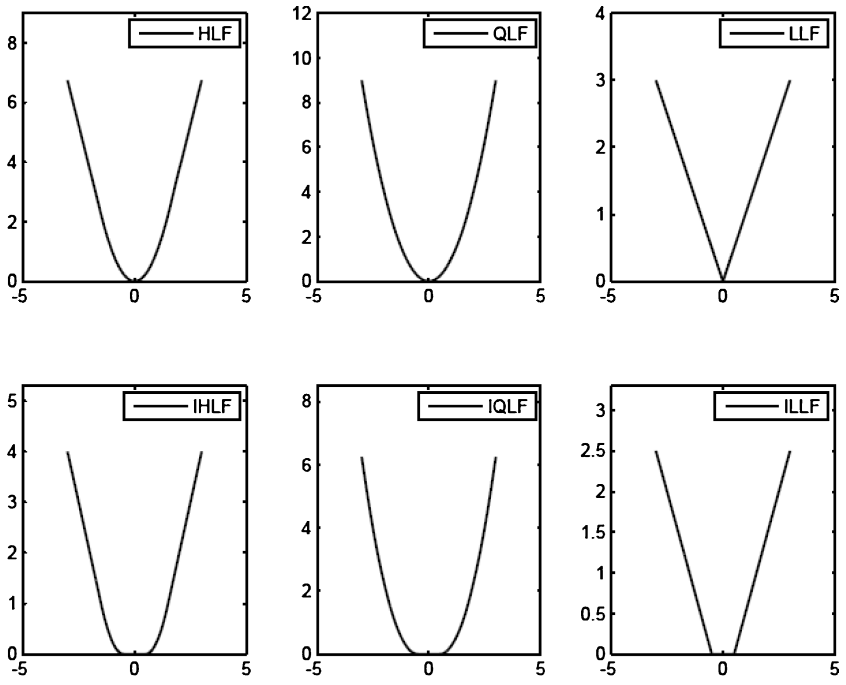

Figure 2: Loss functions derived from equation 3.1.

Let the residual vector be

$$\begin{cases} \mathbf{r}(\boldsymbol{\beta}) = \mathbf{K}\boldsymbol{\beta} - \mathbf{y} \\ r_i(\boldsymbol{\beta}) = \mathbf{K}_i\boldsymbol{\beta} - y_i \end{cases} . \tag{3.4}$$

Defining the sign vector $\mathbf{s}(\boldsymbol{\beta}) = [s_1(\boldsymbol{\beta}), \cdots, s_n(\boldsymbol{\beta})]^T$ by

$$s_i(\boldsymbol{\beta}) = \begin{cases} 1 & \mathit{if}\ \varepsilon < r_i(\boldsymbol{\beta}) < \Delta \\ -1 & \mathit{if}\ -\Delta < r_i(\boldsymbol{\beta}) < -\varepsilon \\ 0 & \text{otherwise} \end{cases} , \tag{3.5}$$

the sign vector $\bar{\mathbf{s}}(\boldsymbol{\beta}) = [\bar{s}_1(\boldsymbol{\beta}), \cdots, \bar{s}_n(\boldsymbol{\beta})]^T$ by

$$\bar{s}_i(\boldsymbol{\beta}) = \begin{cases} 1 & \mathit{if}\ r_i(\boldsymbol{\beta}) \geq \Delta \\ -1 & \mathit{if}\ r_i(\boldsymbol{\beta}) \leq -\Delta \\ 0 & \text{otherwise} \end{cases} , \tag{3.6}$$

and the active matrix

$$\mathbf{W}(\boldsymbol{\beta}) = diag\left\{w_1(\boldsymbol{\beta}), \cdots, w_n(\boldsymbol{\beta})\right\} \tag{3.7}$$

by $w_i(\boldsymbol{\beta}) = s_i^2(\boldsymbol{\beta})$, we can rewrite equation 3.3 as

$$\min_{\boldsymbol{\beta}}\left(L_{\varepsilon,\Delta}(\boldsymbol{\beta}) = \sum_{i=1}^{n} w_i(\boldsymbol{\beta})(|r_i(\boldsymbol{\beta})| - \varepsilon)^2 + \sum_{i=1}^{n} \bar{s}_i^2(\boldsymbol{\beta})(\Delta - \varepsilon)\left(2\left|r_i(\boldsymbol{\beta})\right| - \Delta - \varepsilon\right) + \lambda\boldsymbol{\beta}^T\mathbf{K}\boldsymbol{\beta}\right). \tag{3.8}$$

Rearranging equation 3.8, we can obtain a more compact expression:

$$\min_{\boldsymbol{\beta}}\begin{pmatrix} L_{\varepsilon,\Delta}(\boldsymbol{\beta}) = \mathbf{r}(\boldsymbol{\beta})^T\mathbf{W}(\boldsymbol{\beta})\mathbf{r}(\boldsymbol{\beta}) - 2\varepsilon\mathbf{r}(\boldsymbol{\beta})^T\mathbf{s}(\boldsymbol{\beta}) + 2(\Delta - \varepsilon)\mathbf{r}(\boldsymbol{\beta})^T\bar{\mathbf{s}}(\boldsymbol{\beta}) \\ +\lambda\boldsymbol{\beta}^T\mathbf{K}\beta + \varepsilon^T\mathbf{W}(\boldsymbol{\beta})\varepsilon - \left(\Delta^2 - \varepsilon^2\right)\bar{\mathbf{s}}(\boldsymbol{\beta})^T\bar{\mathbf{s}}(\boldsymbol{\beta}) \end{pmatrix}. \tag{3.9}$$

Let us consider some basic properties of $L_{\varepsilon,\Delta}(\boldsymbol{\beta})$. First, $L_{\varepsilon,\Delta}(\boldsymbol{\beta})$ is a piecewise quadratic function. Second, $L_{\varepsilon,\Delta}(\boldsymbol{\beta})$ is a convex function, so it has a unique minimizer. Finally, $L_{\varepsilon,\Delta}(\boldsymbol{\beta})$ is continuously differentiable with respect to $\boldsymbol{\beta}$. Although $L_{\varepsilon,\Delta}(\boldsymbol{\beta})$ is not twice differentiable, one still can use the finite Newton algorithm by defining the generalized Hessian matrix. The flowchart of the finite Newton algorithm is as follows:

Algorithm 3.1: IHLF-SVR-FN

```
1. Choose a suitable starting point $\boldsymbol{\beta}^0$ and set $k = 0$;
2. Check whether $\boldsymbol{\beta}^k$ is the optimal solution of equation 3.9. If so, stop;
3. Compute the Newton step $\mathbf{h}$;
4. Choose the step size $t$ by the exact line search. Set $\boldsymbol{\beta}^{k+1} = \boldsymbol{\beta}^k + t\mathbf{h}$ and
   $k = k + 1$.
```

4 Practical Implementation

4.1 Computing the Newton Step. For any given value of $\boldsymbol{\beta}$, we say that a point $\mathbf{x}_i$ is support vector if $\left|r_i(\boldsymbol{\beta})\right| > \varepsilon$. Let $sv_1 = \left\{i \,|\, s_i\left(\boldsymbol{\beta}^k\right) \neq 0\right\}$ denote the index set of the support vectors lying in the quadratic part of the loss function, $sv_2 = \left\{i \,|\, \bar{s}_i\left(\boldsymbol{\beta}^k\right) \neq 0\right\}$ the index set of the support vectors lying in the linear part of the loss function, and nsv the index set of the nonsupport vectors. Chapelle (2006) has shown that in support vector classification, the Newton step can be computed at a cost of $O\left(n_{sv_1}^3\right)$ rather than $O\left(n^3\right)$, where

n_{sv_1} denotes the number of the support vectors lying in the quadratic part of the loss function. In the following, we will show that a similar conclusion holds for SVR.

The gradient of $L_{\varepsilon,\Delta}(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ is

$$\begin{aligned}\nabla L_{\varepsilon,\Delta}(\boldsymbol{\beta}) = {} & 2\mathbf{K}^T\mathbf{W}(\boldsymbol{\beta})\mathbf{r}(\boldsymbol{\beta}) - 2\varepsilon\mathbf{K}^T\mathbf{s}(\boldsymbol{\beta}) \\ & + 2(\Delta-\varepsilon)\mathbf{K}^T\bar{\mathbf{s}}(\boldsymbol{\beta}) + 2\lambda\mathbf{K}\boldsymbol{\beta}.\end{aligned} \tag{4.1}$$

Define the set A by

$$A = \left\{\boldsymbol{\beta} \in R^n \;|\exists i,\; \left|r_i(\boldsymbol{\beta})\right| = \varepsilon \;\; or \;\; \left|r_i(\boldsymbol{\beta})\right| = \Delta\right\}. \tag{4.2}$$

The Hessian exists for $\boldsymbol{\beta} \notin A$. For $\boldsymbol{\beta} \in A$, it is (arbitrarily) defined to one of its limits. Thus, we have the generalized Hessian,

$$\nabla^2 L_{\varepsilon,\Delta}(\boldsymbol{\beta}) = 2\mathbf{K}^T\mathbf{W}(\boldsymbol{\beta})\mathbf{K} + 2\lambda\mathbf{K}. \tag{4.3}$$

The Newton step at the kth iteration is given by

$$\begin{aligned}\mathbf{h} &= -\left(\nabla^2 L_{\varepsilon,\Delta}\left(\boldsymbol{\beta}^k\right)\right)^{-1}\nabla L_{\varepsilon,\Delta}\left(\boldsymbol{\beta}^k\right) \\ &= \left(\mathbf{W}\left(\boldsymbol{\beta}^k\right)\mathbf{K} + \lambda\mathbf{I}\right)^{-1}\left(\mathbf{W}\left(\boldsymbol{\beta}^k\right)\mathbf{y} + \varepsilon\mathbf{s}\left(\boldsymbol{\beta}^k\right) - (\Delta-\varepsilon)\bar{\mathbf{s}}\left(\boldsymbol{\beta}^k\right)\right) - \boldsymbol{\beta}^k,\end{aligned} \tag{4.4}$$

where $\mathbf{I}$ is the identity matrix. For convenience of expression, we reorder the training samples such that the first n_{sv_1} training samples are the support vectors lying in the quadratic part of the loss function and the training samples from $n_{sv_1}+1$ to $n_{sv_1}+n_{sv_2}$ are the support vectors lying in the linear part of the loss function. Thus, equation 4.4 can be rewritten as

$$\begin{aligned}\begin{bmatrix}\mathbf{h}_{sv_1}\\ \mathbf{h}_{sv_2}\\ \mathbf{h}_{nsv}\end{bmatrix} = {} & \begin{bmatrix}\mathbf{K}_{sv_1,sv_1}+\lambda\mathbf{I}_{sv_1,sv_1} & \mathbf{K}_{sv_1,sv_2} & \mathbf{K}_{sv_1,nsv}\\ \mathbf{0} & \lambda\mathbf{I}_{sv_2,sv_2} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \lambda\mathbf{I}_{nsv,nsv}\end{bmatrix}^{-1} \\ & \times\begin{bmatrix}\mathbf{y}_{sv_1}+\varepsilon\left[\mathbf{s}\left(\boldsymbol{\beta}^k\right)\right]_{sv_1}\\ -(\Delta-\varepsilon)\left[\bar{\mathbf{s}}\left(\boldsymbol{\beta}^k\right)\right]_{sv_2}\\ \mathbf{0}\end{bmatrix} - \begin{bmatrix}\boldsymbol{\beta}^k_{sv_1}\\ \boldsymbol{\beta}^k_{sv_2}\\ \boldsymbol{\beta}^k_{nsv}\end{bmatrix}.\end{aligned} \tag{4.5}$$

Together with the matrix inversion result from Zhang (2004), we can simplify equation 4.5 as

$$\begin{bmatrix} \mathbf{h}_{sv_1} \\ \mathbf{h}_{sv_2} \\ \mathbf{h}_{nsv} \end{bmatrix} = \begin{bmatrix} (\mathbf{K}_{sv_1,sv_1} + \lambda \mathbf{I}_{sv_1,sv_1})^{-1} \left(\mathbf{y}_{sv_1} + \varepsilon \left[\mathbf{s}\left(\boldsymbol{\beta}^k\right)\right]_{sv_1} \right. \\ \left. \qquad + \dfrac{(\Delta - \varepsilon)\, \mathbf{K}_{sv_1,sv_2} \left[\bar{\mathbf{s}}\left(\boldsymbol{\beta}^k\right)\right]_{sv_2}}{\lambda} \right) - \boldsymbol{\beta}^k_{sv_1} \\ \dfrac{-(\Delta - \varepsilon)\left[\bar{\mathbf{s}}\left(\boldsymbol{\beta}^k\right)\right]_{sv_2}}{\lambda} - \boldsymbol{\beta}^k_{sv_2} \\ -\boldsymbol{\beta}^k_{nsv} \end{bmatrix}. \tag{4.6}$$

Equation 4.6 indicates that we can compute the Newton step at a cost of $O\left(n^3_{sv_1}\right)$. Instead of inverting the matrix $(\mathbf{K}_{sv_1,sv_1} + \lambda \mathbf{I}_{sv_1,sv_1})$, we can compute the first part of the Newton step by solving the positive definite system:

$$(\mathbf{K}_{sv_1,sv_1} + \lambda \mathbf{I}_{sv_1,sv_1})\, \mathbf{h}_{sv_1} = \mathbf{y}_{sv_1} + \varepsilon \left[\mathbf{s}\left(\boldsymbol{\beta}^k\right)\right]_{sv_1} + \frac{(\Delta - \varepsilon)\, \mathbf{K}_{sv_1,sv_2} \left[\bar{\mathbf{s}}\left(\boldsymbol{\beta}^k\right)\right]_{sv_2}}{\lambda}. \tag{4.7}$$

Many methods, such as gaussian elimination, conjugate gradient, and factorization, can be used to find the solution to equation 4.7. In the current implementation, we solve equation 4.7 using Matlab command "\" that employs Cholesky factorization.

4.2 Exact Line Search. The minimizer of a piecewise-smooth, convex quadratic function can be found by identifying the points at which the second derivative jumps, sorting these points, and successively searching over those points until a point where the slope changes sign is located. (For more details, refer to Madsen & Nielsen, 1990, and Keerthi & DeCoste, 2005.) The complexity of this line search is $O(m \log(m))$. Since the Newton step is much more expensive, the line search does not add to the complexity of the algorithm.

4.3 Initialization. The initialization of IHLF-SVR-FN is an important problem. If we use a randomly generated $\boldsymbol{\beta}$ as a starting point, we would have to invert an $n \times n$ matrix at the first Newton step. But if we can identify the support vectors ahead of time, we need to invert only an $n_{sv_1} \times n_{sv_1}$ matrix at the first Newton step.

According to Chapelle's suggestion (Chapelle, 2006), we can identify the support vectors by a recursive procedure. We start from a small number of training samples, train, double the number of training samples, retrain, and so on. In this way, the support vectors are rather well identified, avoiding the inversion of the $n \times n$ matrix. To distinguish this algorithm from the original finite Newton algorithm, we call it a recursive finite Newton algorithm (IHLF-SVR-RFN).

4.4 Checking Convergence. Once the support vectors remain unchanged, IHLF-SVR-FN has found the optimal solution. We use this property to check the convergence of IHLF-SVR-FN.

4.5 Finite Convergence and Computational Complexity

Theorem 1: *IHLF-SVR-FN terminates at the global minimum solution of equation 3.9 after a finite number of iterations.*

The proof of theorem 1 is very much along the lines of the proof of theorem 1 in Keerthi and DeCoste (2005), which is easily extended to a piecewise quadratic loss function, although it considers only a squared hinge loss function.

In IHLF-SVR-RFN, the most time-consuming step is computing the Newton step, whose complexity is $O(n(n_{sv_1} + n_{sv_2}) + n_{sv_1}^3)$, where n_{sv_2} denotes the number of the support vectors lying in the linear part of the loss function. The first term is the cost of finding the support vectors and the second term the cost of solving the linear system, equation 4.7. The number of iterations, that is, the loops of steps 2 to 4, is usually small, say, 5 to 30.

5 Experiments

In order to verify the effectiveness of IHLF-SVR-RFN and IQLF-SVR-RFN (the insensitive quadratic loss function is used), we compare them with LIBSVM 2.82 on some benchmark data sets. LIBSVM 2.82 is the fastest version of LIBSVM, where the second-order information is used to select the working set. Gaussian kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_2^2)$ is used to construct nonlinear SVR. All the experiments are run on a personal computer with 2.4 GHz processors, 1 GB memory, and Windows XP operation systems. (Matlab code for IHLF-SVR-RFN is available online at http://see.xidian.edu.cn/graduate/lfbo/.)

We use the data sets Abalone, Computer Activity, and House16H (Rasmussen et al., 1996) for our comparisons. The Abalone data set consists of 4177 samples with 8 attributes. The task is to predict the age of abalone from all eight physical measurements. The Computer Activity data set was collected from a Sun Sparcstation 20/712 with 128 Mb of memory running in a multiuser university department. It consists of 8192 samples with 21 attributes. The task is to predict the portion of time that CPUs run in user mode from all 21 attributes. The House16H data set consists of 22,784

Table 1: Training Time on the Abalone, Computer Activity, and House16H Data Sets.

	IHLF-SVR-RFN	IQLF-SVR-RFN	LIBSVM 2.82	ε	Δ
Abalone	2.48	4.30	5.99	0.100	0.110
Computer	11.97	18.43	22.48	0.050	0.055
House16H	1640.81	/	653.05	0.050	0.060

samples with 16 attributes. The task is to predict the median price of the houses in a small survey region.

5.1 Comparison with LIBSVM 2.82. The Abalone data set is randomly split into 3000 training samples and 1177 test samples, the Computer Activity data set into 5000 training samples and 4192 test samples, and the House16H data set into 20,000 training samples and 2784 test samples. For each training-test pair, the training samples are scaled into the interval $[-1, 1]$, and the test samples are adjusted using the same linear transformation.

For the Abalone and Computer Activity data sets, we precompute the entire kernel matrix for all three algorithms. Since the value of the parameter pair (γ, C) that optimizes the generalization performance is usually not known prior, the value that is ultimately used to design SVR is usually determined by trying many different values of (γ, C). Thus, it is important to understand how different values, optimal and otherwise, affect training time.

Four experiments are performed on the Abalone and Computer Activity data sets. In the first experiment, we fix the values of ε and Δ shown in Table 1 and search the optimal parameter pair (γ^*, C^*) from the set $\{2^{-4}/d, 2^{-3}/d, \cdots, 2^3/d, 2^4/d\} \times \{2^{-3}, 2^{-2}, \cdots, 2^7, 2^8\}$, which gives the best generalization performance on the test samples, where d is the number of the attributes of samples. Therefore, for each problem, we try $9 \times 12 = 108$ combinations. The mean training time of the three algorithms on all parameter pairs is reported in Table 1. In the second experiment, we fix the value of the kernel parameter and investigate the robustness of the three algorithms against the variation of the regularization parameter C. In the third experiment, we fix the value of the regularization parameter and investigate the robustness of the three algorithms against the variation of the kernel parameter γ. In the fourth experiment, we investigate the influence of training samples size on the training time of the three algorithms. The values of the kernel parameter and the regularization parameter are set to (γ^*, C^*).

From Table 1, we can see that IHLF-SVR-RFN outperforms IQLF-SVR-RFN and LIBSVM 2.82 in terms of the training time. From Figure 3, we observe that the training time of LIBSVM 2.82 has a sharp increase for large

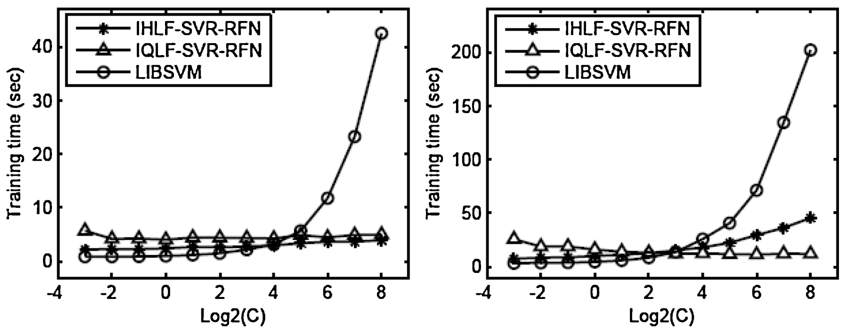

Figure 3: Variation of the training time with the regularization parameter on the Abalone (left) and Computer Activity (right) data sets. γ is fixed to $2^2/d$.

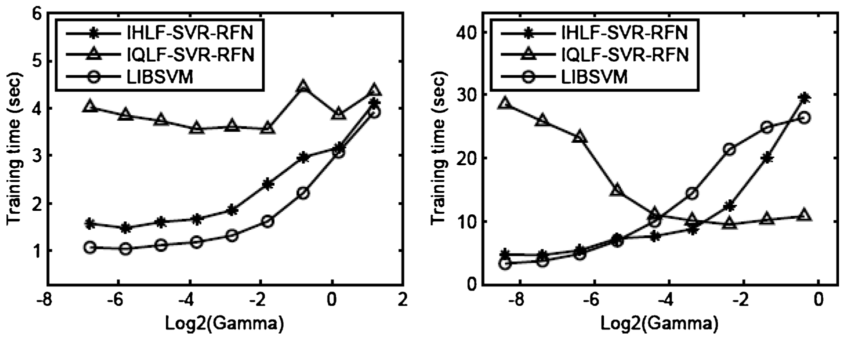

Figure 4: Variation of the training time with the kernel parameter on the Abalone (left) and Computer Activity (right) data sets. For the Abalone data set, C is fixed to 2^3, and for the Computer Activity data set, C is fixed to 2^4.

C value, and hence IHLF-SVR-RFN and IQLF-SVR-RFN are much more robust against variation of C than LIBSVM 2.82. From Figure 4, we observe that the training time of IHLF-SVR-RFN and LIBSVM 2.82 increases with an increasing kernel parameter. From Figure 5, we observe that in terms of the training time IHLF-SVR-RFN and IQLF-SVR-RFN are superior to LIBSVM 2.82 on the Abalone data set and inferior to LIBSVM 2.82 on the Computer Activity data set using the optimal parameter pair.

For the House16H data sets, it is impossible to store the entire kernel matrix because of insufficient memory, so we need to compute the kernel matrix in each Newton step. Since it is computationally prohibitive to search the set of grid values using all the training samples, we use a random subset of the training samples of size 5000 to choose (γ^*, C^*). The training time of IHLF-SVR-RFN and LIBSVM 2.82 on (γ^*, C^*) is given in Table 1. Note that

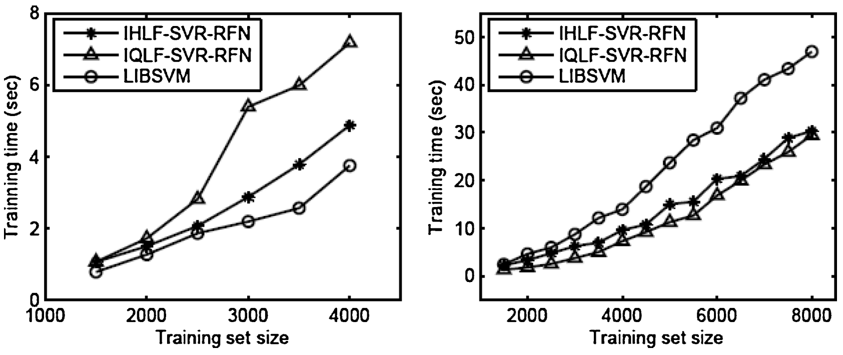

Figure 5: Variation of the training time with the training set size on the Abalone (left) and Computer Activity (right) data sets. For the Abalone data set, (γ, C) is fixed to $(2^2/d, 2^3)$, and for the Computer Activity data set, (γ, C) is fixed to $(2^2/d, 2^4)$.

IQLF-SVR-RFN cannot run on this data set because of insufficient memory. From Table 1, we can see that LIBSVM 2.82 is faster than IHLF-SVR-RFN on this data set. The main reason is that IHLF-SVR-RFN needs to reevaluate the $n \times (n_{sv_1} + n_{sv_2})$ kernel matrix in each Newton step. However, we believe that IHLF-SVR-RFN has the advantage for large-scale optimization because the computation of the kernel matrix can be easily parallelized, which could improve the training time significantly.

5.2 Influence of Δ. The purpose of the experiments in this section is to study the influence of Δ on test error and training time. The partition of the training samples and test samples is the same as in section 5.1. The values of the kernel parameter and the regularization parameter are set to (γ^*, C^*), and the value of ε is the same as in Table 1. The final errors are averaged over 10 random splits of the full data sets. Figures 6 and 7 show the variation of the test error and the training time with Δ on the Abalone and Computer Activity data sets, respectively.

From Figure 6, we observe that the performance of IHLF-SVR-RFN is close to that of LIBSVM 2.82 (the insensitive linear loss function is used) when Δ approaches ε; the performance of IHLF-SVR-RFN is close to that of IQLF-SVR-RFN when Δ is significantly greater than ε. This is consistent with the discussion in section 3. From Figure 7, we observe that the training time of IHLF-SVR-RFN fluctuates with the variation of Δ value. The main reason is that the training time is affected by the number of support vectors and the Newton step. The former often increases with an increasing Δ, and hence some training time is added for large Δ value; however, the latter often decreases with an increasing Δ, and hence some training time is subtracted for large Δ value.

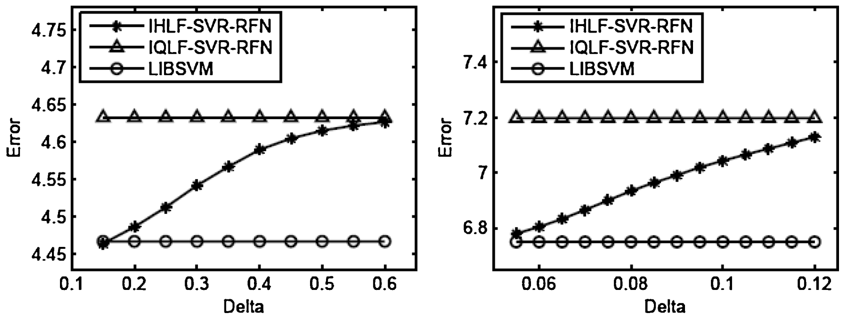

Figure 6: Influence of Δ on the test error on the Abalone (left) and Computer Activity (right) data sets.

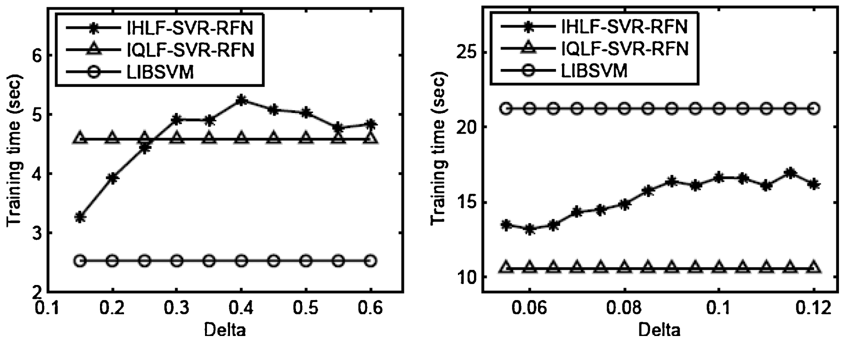

Figure 7: Influence of Δ on the training time on the Abalone (left) and Computer Activity (right) data sets.

6 Conclusion and Discussions

In this letter, we present a recursive finite Newton algorithm for training nonlinear SVR. First, we propose the insensitive Huber loss function and show that several popular loss functions are special cases. Then we discuss in detail the implementation of IHLF-SVR-RFN. Finally, we verify the effectiveness of the proposed algorithm on three benchmark regression data sets.

The computational complexity of IHLF-SVR-RFN can be decreased by some approximation strategies. For example, if the kernel matrix is sparse, the Newton step can be solved more efficiently. We can also approximate the objective function 3.9 by a matching pursuit approach. Keerthi, Chapelle, and DeCoste (2006) implemented that in the classification case, and it is straightforward to generalize it to the regression case.

Acknowledgments

We thank the two reviewers for their helpful comments, which greatly improved this letter, and Rainer Stollhoff and Bruce Rogers for their help in proofreading the manuscript. This work was supported by the National Natural Science Foundation of China under grant 60372050 and the National Defense Preresearch Foundation of China under Grant A1420060172.

References

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

Chapelle, O. (2006). *Training a support vector machine in the primal*. (MPI-Tech. Rep. no. 147). Tübingen: Max Planck Institute for Biological Cybernetics.

Clarke, F. H. (1983). *Optimization and nonsmooth analysis*. New York: Wiley.

Fan, R. E., Chen P. H., & Lin C. J. (2005). Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research, 6*, 1889–1918.

Fung, G., & Mangasarian, O. L. (2003). Finite Newton method for Lagrangian support vector machine classification. *Neurocomputing, 55*(1–2), 39–55.

Hiriart-Urruty, J. B., Strodiot, J. J., & Nguyen V. H. (1984). Generalized Hessian matrix and second-order optimality conditions for problems with CL1 data. *Applied Mathematics and Optimization, 11*, 43–56.

Huber, P. (1981). *Robust statistics*. New York: Wiley.

Joachims, T. (1999). Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods—Support vector learning*. Cambridge, MA: MIT Press.

Keerthi, S. S., Chapelle, O., & DeCoste D. (2006). Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research, 7*, 1493–1515.

Keerthi, S. S., & DeCoste D. M. (2005). A modified finite Newton method for fast solution of large scale linear SVMS. *Journal of Machine Learning Research, 6*, 341–361.

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., & Murthy, K. R. K. (2001). Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation, 13*(3), 637–649.

Kimeldorf, G. S., & Wahba G. (1970). A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *Annals of Mathematical Statistics, 41*, 495–502.

Madsen, K., & Nielsen H. B. (1990). Finite algorithms for robust linear-regression. *BIT, 30*(4), 682–699.

Mangasarian, O. L. (2002). A finite Newton method for classification. *Optimization Methods and Software, 17*(5), 913–929.

Platt, J. (1999). Sequential minimal optimization: A fast algorithm for training support vector machines. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods—Support vector learning*. Cambridge, MA: MIT Press.

Rasmussen, C., Neal, R., Hinton G., van Camp D., Ghahramani Z., Kustra, R., & Tibshirani R. (1996). *The DELVE manual*. Available online at http://www.cs.toronto.edu/~delve.

Shevade, S. K., Keerthi, S. S., Bhattacharyya, C., & Murthy, K. R. K. (2000). Improvements to the SMO algorithm for SVM regression. *IEEE Transactions on Neural Networks, 11*(5), 1188–1193.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on SVR. *Statistics and Computing, 14*(3), 199–222.

Vapnik, V. (1998). *Statistical learning theory*, New York: Wiley.

Zhang, X. D. (2004). *Matrix analysis and application*. New York: Springer.

Received January 22, 2006; accepted July 19, 2006.