Neural Computing and Applications (2020) 32:11285–11309
https://doi.org/10.1007/s00521-019-04625-8

ORIGINAL ARTICLE

Robust twin support vector regression based on Huber loss function

S. Balasundaram[1] · Subhash Chandra Prasad[1]

Received: 12 September 2018 / Accepted: 22 November 2019 / Published online: 13 December 2019

Abstract
Construction of robust regression learning models to fit data with noise is an important and challenging problem of data regression. One of the ways to tackle this problem is the selection of a proper loss function showing insensitivity to noise present in the data. Since Huber function has the property that inputs with large deviations of misfit are penalized linearly and small errors are squared, we present novel robust regularized twin support vector machines for data regression based on Huber and ε-insensitive Huber loss functions in this study. The proposed regression models result in solving a pair of strongly convex minimization problems in simple form in primal whose solutions are obtained by functional and Newton–Armijo iterative algorithms. The finite convergence of Newton–Armijo algorithm is proved. Numerical tests are performed on noisy synthetic and benchmark datasets, and their results are compared with few popular regression learning algorithms. The comparative study clearly shows the robustness of the proposed regression methods and further demonstrates their effectiveness and suitability.

Keywords Huber M-estimator · Kernel methods · Regression · Support vector machine

1 Introduction

Support vector machines (SVMs) are powerful machine learning tools for data classification and regression problems. Because of their superior generalization performance over other machine learning methods, such as the artificial neural networks (ANNs), they have been successfully applied on a variety of real-world problems such as image processing, bioinformatics and financial regression [16, 26, 27].

The basic idea of SVM is in determining the optimal separating hyperplane by maximizing the margin between two parallel hyperplanes, leading to solving a quadratic programming problem (QPP) [11, 36]. Although SVM owns better generalization performance than other learning methods, its training time complexity is $O(m^3)$, where m is the total number of training samples. To overcome this challenge, by considering the proximity of the data to one of the two nonparallel hyperplanes, a nonparallel hyperplane classifier called generalized eigenvalue proximal support vector machine (GEPSVM) was first proposed in Mangasarian and Wild [25]. Similar in spirit, Jayadeva et al. [22] proposed a twin support vector machine (TWSVM) wherein two nonparallel hyperplanes are constructed so that each plane is closer to one class of data points but as far away as possible from the data points of other class. Such strategy leads to solving two QPPs of smaller size which makes the training of TWSVM four times faster than the standard SVM [22]. Due to its low computational training cost, TWSVM attracted lot of interest in the literature [2, 29, 38].

With the introduction of ε-insensitive loss function [36], support vector machine method for data regression has been proposed on the principle of SVM. In fact, in the classical support vector regression method (ε-SVR), a regression function $f(\mathbf{x})$ is determined such that data points outside of the ε-tube between $f(\mathbf{x}) - \varepsilon$ and $f(\mathbf{x}) + \varepsilon$ contribute to the error and at the same time $f(\mathbf{x})$ is made as flat as possible. Recently, in the spirit of TWSVM, Peng [28] proposed twin support vector machine for data regression (TSVR) where a pair of ε-insensitive down-bound and up-bound functions are constructed by solving two smaller

✉ S. Balasundaram
balajnu@gmail.com; bala0400@mail.jnu.ac.in

Subhash Chandra Prasad
subhas15_scs@jnu.ac.in

[1] School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi 110067, India

QPPs, whereas ε-SVR solves a larger QPP. This makes the learning speed of TSVR significantly faster than ε-SVR [28]. In [32], a ε-insensitive twin support vector regression (ε-TSVR) algorithm is proposed where the structural risk is minimized by adding a regularization term resulting in better generalization performance and training time than ε-SVR. With the aim of estimating flexible insensitive zone with high sparsity and better generalization capability, pairing support vector regression (PSVR) is proposed recently and the interested reader is referred to [17]. On the study of some of the interesting twin SVR models reported in the literature, we refer to Balasundaram and Gupta [1], Balasundaram and Tanveer [3], Peng et al. [30], Rastogi et al. [31], Yang et al. [39].

Learning to fit data with noise/outliers is an important and challenging research problem. The data arising from real-world applications are in general subject to the presence of noise and outliers. When the observed data are noisy, the learning model may try to fit the corrupt data, which often results in poor generalization performance in the test phase [10]. In a robust regression model, if the estimated error of a data point is large, then by considering it as an outlier, its contribution to learning can be reduced. Several robust regression models have been reported in the literature to reduce the negative effect of outliers [6, 8–10, 37, 40, 41].

One of the approaches proposed in the literature to achieve robustness is the elimination of the outliers from the training set by applying an outlier detection technique and training the remaining set of inputs with a learning model [9]. The main difficulty of this approach is in identifying the outliers, especially when no prior knowledge about error distribution is available [6]. Another drawback is the increase in computational cost.

The other approach in designing robust regression models is in applying a suitable loss function which can resist the effect of outliers [44]. For example, in Chuang et al. [10], a robust SVR is proposed where the ε-insensitive loss function is replaced by a robust tanh estimation function. As we know, quadratic loss function, 1-norm loss function and ε-insensitive loss function are some of the empirical risk functions commonly used in regression problems [11, 36]. Quadratic loss function is smooth and hence attractive but it is non robust. Though both 1-norm and ε-insensitive loss functions are considerably less sensitive to large error of misfit than quadratic loss function, they are only continuous which precludes the application of popular numerical minimization methods of solving.

In recent times, Huber function [20] is applied as a robust error measure to handle noise/outliers. Since samples with large deviations are penalized linearly like 1-norm loss function and, however, small errors are squared like quadratic loss function, it is a hybrid function and is considerably insensitive to large noise. Moreover, Huber function is convex and, unlike 1-norm loss and ε-insensitive loss functions, it is differentiable everywhere. Because of differentiability, it is reasonable to suppose that Huber SVR will be easier to minimize than 1-norm SVR and ε-SVR and in addition it will be possible to apply gradient-based optimizers. In Guitton and Symes [15], as a robust model, it is proposed to minimize the Huber regression problem as a function of the residuals by a quasi-Newton method and successfully applied on an inverse problem for velocity analysis. A robust regression model for Bayesian support vector regression is constructed in Chu et al. [8] where the Huber and ε-insensitive loss functions are combined into a unified function to become ε-insensitive Huber function. As hybrid approaches, robust regularized kernel regression for Huber and similarly ε-insensitive Huber loss functions are proposed in Zhu et al. [44]. Chen et al. [7] proposed a robust algorithm of SVR in primal with trimmed Huber loss function claiming high robustness to outliers. In Mangasarian and Musicant [24], finding an approximate solution of the regression problem as a function of the residuals via robust Huber M-estimator is proposed to become an equivalently solvable convex quadratic problem. By assigning small weights to samples with large error, it proposed that the effect of noise can be reduced [34]. Also some researchers employed nonconvex loss functions instead of convex functions to obtain robustness to outliers [42, 43]. The main drawbacks of the previously studied robust Huber regression models are that they are significantly complex and require special care in designing algorithms for solving them. Recently, robust support vector regression models in simple form have been proposed in Balasundaram and Meena [4] by defining the misfit error via asymmetric Huber and ε-insensitive asymmetric Huber functions.

The goal of this paper is to present robust, regularized, Huber and ε-insensitive Huber twin support vector machine formulations for data regression in simple form whose solutions can be obtained by simple, well-known iterative methods. Our proposed work is an extension of Shao et al. [32] on ε-insensitive twin support vector regression and the study of Balasundaram and Meena [4] on robust Huber and ε-insensitive Huber SVR leading to minimization problems in simple form. From the previous study [5] that primal approaches usually are superior to dual approaches, it is proposed to solve the minimization problems directly in primal.

In this work, all vectors are considered as column vectors. For a vector $\mathbf{x} = (x_1, \ldots, x_n)^t \in R^n$, its transpose and 2-norm will be denoted by $\mathbf{x}^t$ and $||\mathbf{x}||$, respectively. We define the plus function $\mathbf{x}_+$ by: $(\mathbf{x}_+)_i = \max\{0, x_i\}$ where $i = 1, \ldots, n$. The m-dimensional column vector of zeros

and similarly the vector of ones will be denoted by $\mathbf{0}$ and $\mathbf{e}$, respectively, and the identity matrix of appropriate size will be denoted by I.

The paper is organized as follows. As related work, Sect. 2 provides a brief review on the formulations of ε-insensitive support vector regression (ε-SVR), least-squares support vector regression (LS-SVR) and twin support vector regression (TSVR). Section 3 presents novel robust twin support vector regression models using Huber and ε-insensitive Huber loss functions in primal whose solutions are obtained by functional and Newton–Armijo iterative methods. For comparison purpose, numerical tests on (1) synthetic datasets with different types of noise/outliers and (2) benchmark datasets are performed in Sect. 4, while 5 concludes the paper.

2 Related work

2.1 ε-Insensitive support vector regression

In this subsection, we briefly describe the formulation of support vector regression (SVR) with ε-insensitive loss function (ε-SVR) proposed by Vapnik [36].

Suppose we are given a set of training examples $\{(\mathbf{x}_i, y_i)\}_{i=1,2,\ldots,m}$ such that for each input $\mathbf{x}_i \in R^n$, let $y_i \in R$ be its corresponding observed value. The classical SVR (ε-SVR) aims at searching an optimal function $f(\mathbf{x})$ such that an ε-insensitive tube around the training examples is set within which errors are neglected.

To obtain a nonlinear regression estimation function, the inputs are mapped into a higher dimensional feature space via a nonlinear mapping $\varphi(.)$ and a linear learning regressor is obtained in the feature space [11, 36]. Suppose that the nonlinear regression estimating function $f : R^n \rightarrow R$ takes the form:

$$f(\mathbf{x}) = \varphi(\mathbf{x})^t\mathbf{w} + b \text{ for } \mathbf{x} \in R^n,$$

where $\mathbf{w}$ is a vector in the feature space and b is a scalar threshold.

It is well known that the ε-insensitive SVR formulation leads to solving the following unconstrained optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^t\mathbf{w} + C\sum_{i=1}^{m} |f(\mathbf{x}_i) - y_i|_\varepsilon,$$

where $|f(\mathbf{x}_i) - y_i|_\varepsilon = \max\{0, |f(\mathbf{x}_i) - y_i| - \varepsilon\}$ is the ε-insensitive error at $\mathbf{x_i}$ and $C > 0$, $\varepsilon > 0$ are parameters. By introducing slack variables ξ_{1i} and ξ_{2i}, the primal problem can be reformulated as a constrained minimization problem defined as [11, 36]

$$\min_{\mathbf{w},b,\xi_1,\xi_2} \frac{1}{2}\mathbf{w}^t\mathbf{w} + C\sum_{i=1}^{m} (\xi_{1i} + \xi_{2i}) \tag{1}$$

subject to

$$y_i - \varphi(\mathbf{x}_i)^t\mathbf{w} - b \le \varepsilon + \xi_{1i},$$
$$\varphi(\mathbf{x}_i)^t\mathbf{w} + b - y_i \le \varepsilon + \xi_{2i}$$

and

$$\xi_{1i}, \xi_{2i} \ge 0 \text{ for } i = 1, 2, \ldots, m.$$

By introducing Lagrange multipliers $\mathbf{u}_1 = (u_{11}, \ldots, u_{1m})^t$ and $\mathbf{u}_2 = (u_{21}, \ldots, u_{2m})^t$, the Wolfe dual of (1) will be constructed and the dual problem will be solved. In fact, replacing the dot product $\varphi(\mathbf{x}_i)^t\varphi(\mathbf{x}_j)$ by a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$, the dual of (1) will become:

$$\begin{aligned}\min_{\mathbf{u}_1,\mathbf{u}_2} &\frac{1}{2}\sum_{i,j=1}^{m} (u_{1i} - u_{2i})(u_{1j} - u_{2j})k(\mathbf{x}_i, \mathbf{x}_j) \\ &+\varepsilon\sum_{i=1}^{m} (u_{1i} + u_{2i}) - \sum_{i=1}^{m} y_i(u_{1i} - u_{2i})\end{aligned} \tag{2}$$

subject to

$$\sum_{i=1}^{m} (u_{1i} - u_{2i}) = 0 \text{ and } \mathbf{0} \le \mathbf{u}_1, \mathbf{u}_2 \le C\mathbf{e}.$$

Using the solution of the dual problem (2), the nonlinear fitting function $f(.)$ is obtained as [11, 36] for any $\mathbf{x} \in R^n$,

$$f(\mathbf{x}) = \sum_{i=1}^{m} (u_{1i} - u_{2i})k(\mathbf{x}, \mathbf{x}_i) + b.$$

For more details on SVR, see Cristianini and Shawe-Taylor [11] and Vapnik [36].

2.2 Least-squares support vector regression

In this subsection, we brief on least squares SVR (LS-SVR) method [35]. Instead of considering the inequality constraints in the SVR formulation (1), assuming the equality constraints leads to LS-SVR formulation as a minimization problem

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^t\mathbf{w} + \frac{C}{2}\sum_{i=1}^{m} \xi_i^2$$

subject to

$$y_i = \mathbf{w}^t\varphi(\mathbf{x}_i) + b + \xi_i, \quad i = 1, 2, \ldots, m$$

where the vector $\mathbf{w}$ and the scalar b are the unknowns; $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_m)^t$ is the residue vector; and $C > 0$ is the regularization parameter. By formulating its dual problem and solving it, the solution of LS-SVR is obtained. The main advantage of LS-SVR method is that the unknown

vector variables are determined by solving a system of linear equations. For details, see Suykens et al. [35].

2.3 Twin support vector regression

Motivated by the study of twin support vector machines (TWSVM) [22] for binary classification problem, Peng [28] proposed twin support vector regression (TSVR) algorithm where two nonparallel, ε-insensitive down-bound function $f_1(\mathbf{x})$ and up-bound function $f_2(\mathbf{x})$ are constructed by solving two smaller SVR-type QPPs. This strategy makes the training of TSVR faster than ε-SVR [28].

Suppose the given input data are arranged in a matrix $A \in R^{m\times n}$ in which its ith row becomes the ith training sample $\mathbf{x}_i^t$. Similarly, let $\mathbf{y} \in R^m$ be the vector of observed values whose ith row element is $y_i \in R$. For the kernel function $k(.,.)$ given, the kernel matrix $K(A,A^t)$ of order m is defined such that its ijth element becomes $(K(A,A^t))_{ij} = k(\mathbf{x}_i,\mathbf{x}_j)$. Also for any $\mathbf{x} \in R^n$, let $K(\mathbf{x},A^t) = (k(\mathbf{x},\mathbf{x}_1),\ldots,k(\mathbf{x},\mathbf{x}_m))$ be a row vector.

Then, the nonlinear TSVR aims at finding the ε-insensitive down-bound and up-bound regression functions of the form: for $\mathbf{x} \in R^n$,

$$f_1(\mathbf{x}) = K(\mathbf{x},A^t)\mathbf{w}_1 + b_1 \text{ and } f_2(\mathbf{x}) = K(\mathbf{x},A^t)\mathbf{w}_2 + b_2$$

by solving the following two QPPs [28]

$$\min_{(\mathbf{w}_1,b_1,\xi_1)\in R^{m+1+m}} \frac{1}{2}||\mathbf{y} - \varepsilon_1\mathbf{e} - (K(A,A^t)\mathbf{w}_1 + b_1\mathbf{e})||^2 + C_1\mathbf{e}^t\boldsymbol{\xi}_1$$

subject to

$$\mathbf{y} - (K(A,A^t)\mathbf{w}_1 + b_1\mathbf{e}) \geq \varepsilon_1\mathbf{e} - \boldsymbol{\xi}_1, \quad \boldsymbol{\xi}_1 \geq \mathbf{0}$$

and

$$\min_{(\mathbf{w}_2,b_2,\xi_2)\in R^{m+1+m}} \frac{1}{2}||\mathbf{y} + \varepsilon_2\mathbf{e} - (K(A,A^t)\mathbf{w}_2 + b_2\mathbf{e})||^2 + C_2\mathbf{e}^t\boldsymbol{\xi}_2$$

subject to

$$(K(A,A^t)\mathbf{w}_2 + b_2\mathbf{e}) - \mathbf{y} \geq \varepsilon_2\mathbf{e} - \boldsymbol{\xi}_2, \quad \boldsymbol{\xi}_2 \geq \mathbf{0}$$

respectively, where $C_1, C_2 > 0$; $\varepsilon_1, \varepsilon_2 > 0$ are input parameters and $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2$ are vectors of slack variables.

Once we obtain the pair of solutions $(\mathbf{w}_1, b_1)$ and $(\mathbf{w}_2, b_2)$, the end regression function $f(\mathbf{x})$ is taken to be the mean of $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$, i.e.,

$$f(\mathbf{x}) = \frac{1}{2}(f_1(\mathbf{x}) + f_2(\mathbf{x})) \text{ for all } \mathbf{x} \in R^n.$$

For more details on the problem formulation of TSVR, its method of solution and advantages, the reader is referred to Peng [28].

3 Huber twin support vector regression

Although the popular ε-insensitive loss function introduced by Vapnik [36] is robust and leads to better generalization ability, it can be easily observed that it is only C^0 smooth which precludes the application of popular numerical minimization methods. In this section, we propose novel SVR regression formulations whose loss functions are differentiable everywhere while still robust against large residuals, and further to solve them, we apply the well-known Newton–Armijo algorithm [13, 21] in addition to a simple functional iterative method. With this objective, Huber and $\varepsilon-$ insensitive Huber M-estimator functions [4, 20, 24, 44] are considered as the loss functions to measure the residual error.

Let $L_H(.)$ be the Huber M-estimator, defined by [4, 20, 24, 44]: for $x \in R$,

$$L_H(x) = \begin{cases} x & \text{if } |x| \leq \gamma \\ \gamma(2|x| - \gamma) & \text{if } |x| > \gamma, \end{cases} \tag{3}$$

where $\gamma > 0$ is an input parameter. Clearly, $L_H(.)$ is differentiable everywhere and also is convex. At $x = \pm\gamma$, it switches from quadratic to linear, i.e., it is a hybrid function in which it is quadratic for small errors and linear otherwise. As a function of the residual, the Huber function $L_H(.)$ for different values of γ is illustrated in Fig. 1.

It is worth to extend the Huber function (3) by incorporating a ε-insensitive tube around the inputs and study as an enhanced function. Similar to the soft insensitive loss function introduced in Chu et al. [8], which is a hybrid function resulting a sparse and robust regression model, we consider the ε-insensitive Huber loss function with parameters $\gamma > 0$ and $\varepsilon > 0$, defined by [4, 44]

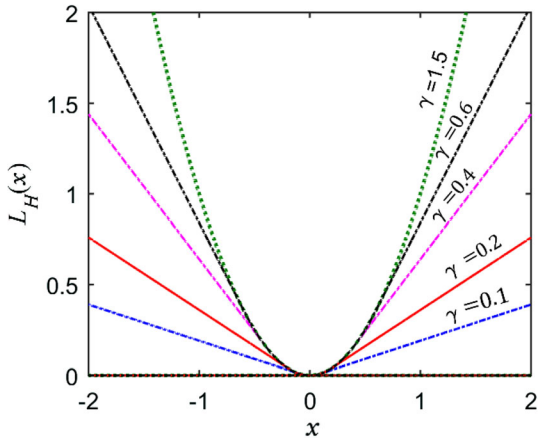

Fig. 1 Illustration of Huber loss function $L_H(x)$ for different values of γ

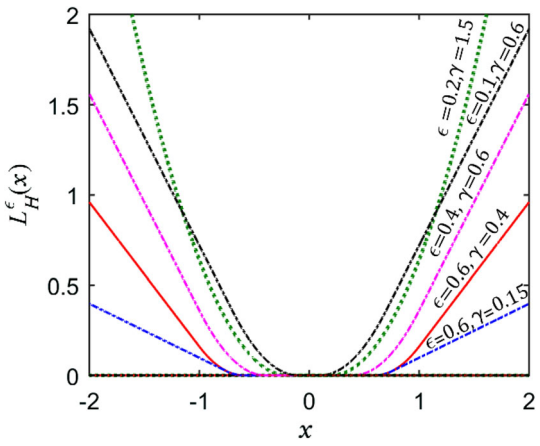

Fig. 2 Illustration of ε-insensitive Huber loss function $L_{\mathrm{H}}^{\varepsilon}(x)$ for different values of ε and γ

$$L_H^{\varepsilon}(x) = \begin{cases} 0 & \text{if } 0 \le |x| < \varepsilon \\ (|x| - \varepsilon)^2 & \text{if } \varepsilon \le |x| < (\varepsilon + \gamma) \\ \gamma(2(|x| - \varepsilon) - \gamma) & \text{if } (\varepsilon + \gamma) \le |x| < \infty, \end{cases} \tag{4}$$

as an enhanced function having the advantages such as insensitivity to outliers and sparseness in its solution representation [8]. Clearly, $L_H^{\varepsilon}(.)$ is symmetric, convex and is a differentiable function having the property that if the difference between the predicted and the observed values falls in the interval $[-\varepsilon, \varepsilon]$, then it will be treated as zero. Again, like in case of (3), the function switches from quadratic to linear at $x = \pm(\varepsilon + \gamma)$, i.e., it is a hybrid function. Finally, when $\varepsilon = 0$, (4) reduces to the Huber M-estimator (3). As a function of the residual, the function $L_H^{\varepsilon}(.)$ for several values of $\gamma > 0$ and $\varepsilon > 0$ is illustrated in Fig. 2.

In Mangasarian and Musicant [24], the problem of finding an approximate solution of the nonlinear regression function defined using kernel of the form:

$$f(\mathbf{x}) = K(\mathbf{x}, A^t)\mathbf{w} + b, \tag{5}$$

is considered for the following generally unsolvable rectangular system of equations

$$f(\mathbf{x}_i) = [K(\mathbf{x}_i, A^t)\ 1]\mathbf{u} = y_i \quad \text{for } i = 1, 2, \ldots, m$$

such that $\mathbf{u} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} \in R^{m+1}$ where $\mathbf{w} \in R^m$ and $b \in R$ are unknowns. Also, for the purpose of reducing the effect of outliers, it is proposed to apply the popular Huber M-estimator (3) for calculating its residual error. This results in the following robust Huber support vector regression (RHSVR) problem defined by

$$\min_{\mathbf{u} \in R^{m+1}} \sum_{i=1}^{m} L_H(f(\mathbf{x}_i) - y_i). \tag{6}$$

Observing that solving problem (6) is fairly complex, it was converted into an equivalent, solvable, unconstrained, convex quadratic problem [24], defined by:

$$\min_{\mathbf{u} \in R^{m+1}, \mathbf{z} \in R^m} \frac{1}{2}\mathbf{z}^t\mathbf{z} + \gamma \sum_{i=1}^{m} |f(\mathbf{x}_i) - y_i - z_i|. \tag{7}$$

Clearly, the proposed formulation (7) introduces much more number of unknown variables than those of (6).

In this work, numerical results obtained by the equivalent RHSVR formulation (7) will be used for comparison with our proposed methods.

Recently, an interesting and rather a much simpler approach in determining the regression function $f(.)$ of form (5) by solving

$$\min_{(\mathbf{w}, b) \in R^{m+1}} \frac{1}{2}(\mathbf{w}^t\mathbf{w} + b^2) + \frac{C}{2}\sum_{i=1}^{m} L(f(\mathbf{x}_i) - y_i)$$

in primal is proposed in Balasundaram and Meena [4] where for any real value x,

$$L(x) = \begin{cases} L_H(x) & \text{for the case of Huber loss} \\ L_H^{\varepsilon}(x) & \text{for the case of } \varepsilon\text{-insensitive Huber loss} \end{cases}$$

Following the approach of ε-twin support vector regression (ε-TSVR) proposed in Shao et al. [32], we now build one-side Huber and ε-insensitive Huber twin SVR models and study their properties and effectiveness.

Let the nonlinear down-bound and up-bound regression functions be defined to be

$$f_1(\mathbf{x}) = K(\mathbf{x}, A^t)\mathbf{w}_1 + b_1 \text{ and } f_2(\mathbf{x}) = K(\mathbf{x}, A^t)\mathbf{w}_2 + b_2, \tag{8}$$

respectively, where $\mathbf{w}_1, \mathbf{w}_2 \in R^m$ and $b_1, b_2 \in R$ are unknowns. Equivalently, by letting $\mathbf{u}_k = \begin{bmatrix} \mathbf{w}_k \\ b_k \end{bmatrix} \in R^{m+1}$ for $k = 1, 2$, the bounding regression functions (8) become

$$f_1(\mathbf{x}) = [K(\mathbf{x}, A^t)\ 1]\mathbf{u}_1 \text{ and } f_2(\mathbf{x}) = [K(\mathbf{x}, A^t)\ 1]\mathbf{u}_2. \tag{9}$$

Using (9), the end regression function $f(\mathbf{x})$ is obtained [28]

$$f(\mathbf{x}) = \frac{f_1(\mathbf{x}) + f_2(\mathbf{x})}{2}. \tag{10}$$

Motivated by ε-TSVR [32], the errors using the one-side Huber loss function corresponding to the down-bound and up-bound regression functions can be computed as

$$\sum_{i=1}^{m} \max\{0, -(y_i - f_1(\mathbf{x}_i))\}^2 - \max\{0, -(y_i - f_1(\mathbf{x}_i) + \gamma)\}^2$$
$$= \sum_{i=1}^{m} [(f_1(\mathbf{x}_i) - y_i)_+]^2 - [(f_1(\mathbf{x}_i) - y_i - \gamma)_+]^2$$
$$= ||(G\mathbf{u}_1 - \mathbf{y})_+||^2 - ||(G\mathbf{u}_1 - \mathbf{y} - \gamma\mathbf{e})_+||^2 \tag{11a}$$

and

$$\sum_{i=1}^{m} \max\{0, -(f_2(\mathbf{x}_i) - y_i)\}^2 - \max\{0, -(f_2(\mathbf{x}_i) - y_i + \gamma)\}^2$$
$$= \sum_{i=1}^{m} [(y_i - f_2(\mathbf{x}_i))_+]^2 - [(y_i - f_2(\mathbf{x}_i) - \gamma)_+]^2$$
$$= ||(\mathbf{y} - G\mathbf{u}_2)_+||^2 - ||(\mathbf{y} - G\mathbf{u}_2 - \gamma\mathbf{e})_+||^2 \tag{11b}$$

respectively, where $G = [K(A, A^t) \quad \mathbf{e}]_{m\times(m+1)}$ is an augmented matrix.

Using an intuitive two-dimensional illustration, we now give the geometric explanation on the misfit error calculation (11a) for one-sided Huber down-bound function $f_1(\mathbf{x})$ which is shown red in color in Fig. 3. Firstly, the whole plane is divided into three parts, i.e., region 1, region 2 and region 3, as illustrated in Fig. 3. Depending on the positioning of the samples, different training errors will be generated. For samples from region 1, $f_1(\mathbf{x}_i) - y_i < 0$ will be true and there is no error. Again, for samples from region 2, $f_1(\mathbf{x}_i) - y_i > 0$ but $f_1(\mathbf{x}_i) - y_i - \gamma < 0$ will be satisfied and in this case the error will become $(f_1(\mathbf{x}_i) - y_i)^2$. Finally, when the samples lie in region 3, $f_1(\mathbf{x}_i) - y_i > 0$ and $f_1(\mathbf{x}_i) - y_i - \gamma > 0$ will be satisfied and for this case $\gamma(2(f(\mathbf{x}_i) - y_i) - \gamma)$ will be the error. Similarly, the misfit error (11b) for the up-bound function $f_2(\mathbf{x})$ is obtained.

By minimizing the sum of the squared error $||G\mathbf{u}_k - \mathbf{y}||^2$, the error using the one-side Huber function and the regularization term $\frac{1}{2}\mathbf{u}_k^t\mathbf{u}_k$, where $k = 1, 2$, our proposed one-side Huber twin SVR (HTSVR) in primal will be formulated, i.e., we solve

$$\min_{\mathbf{u}_1 \in R^{m+1}} L_1(\mathbf{u}_1) = \frac{1}{2}\mathbf{u}_1^t\mathbf{u}_1 + \frac{C_1}{2}||G\mathbf{u}_1 - \mathbf{y}||^2 + \frac{C_3}{2}$$
$$[||(G\mathbf{u}_1 - \mathbf{y})_+||^2 - ||(G\mathbf{u}_1 - \mathbf{y} - \gamma\mathbf{e})_+||^2] \tag{12a}$$

and

$$\min_{\mathbf{u}_2 \in R^{m+1}} L_2(\mathbf{u}_2) = \frac{1}{2}\mathbf{u}_2^t\mathbf{u}_2 + \frac{C_2}{2}||G\mathbf{u}_2 - \mathbf{y}||^2 + \frac{C_4}{2}$$
$$[||(\mathbf{y} - G\mathbf{u}_2)_+||^2 - ||(\mathbf{y} - G\mathbf{u}_2 - \gamma\mathbf{e})_+||^2] \tag{12b}$$

whose solutions will be used to obtain the bound regression functions (9), where $C_1 > 0, C_2 > 0, C_3 > 0$ and $C_4 > 0$ are parameters.

Similarly, the error using one-side $\varepsilon-$ insensitive Huber function corresponding to the down-bound and up-bound regression functions can be computed as

$$\sum_{i=1}^{m} \max\{0, -(y_i - f_1(\mathbf{x}_i)) - \varepsilon\}^2$$
$$- \max\{0, -(y_i - f_1(\mathbf{x}_i) + \varepsilon + \gamma)\}^2$$

and

$$\sum_{i=1}^{m} \max\{0, -(f_2(\mathbf{x}_i) - y_i) - \varepsilon\}^2$$
$$- \max\{0, -(f_2(\mathbf{x}_i) - y_i + \varepsilon + \gamma)\}^2$$

respectively and therefore minimizing the sum of the regularization term and the error terms using the one-side ε-insensitive Huber function and the squared error $||\mathbf{y} - G\mathbf{u}_k||^2$ leads to our proposed one-side ε-insensitive Huber twin SVR (ε-HTSVR) formulation

$$\min_{\mathbf{u}_1 \in R^{m+1}} L_1^{\varepsilon}(\mathbf{u}_1) = \frac{1}{2}\mathbf{u}_1^t\mathbf{u}_1 + \frac{C_1}{2}||G\mathbf{u}_1 - \mathbf{y}||^2 + \frac{C_3}{2}$$
$$[||(G\mathbf{u}_1 - \mathbf{y} - \varepsilon\mathbf{e})_+||^2 - ||(G\mathbf{u}_1 - \mathbf{y} - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+||^2] \tag{13a}$$

and

$$\min_{\mathbf{u}_2 \in R^{m+1}} L_2^{\varepsilon}(\mathbf{u}_2) = \frac{1}{2}\mathbf{u}_2^t\mathbf{u}_2 + \frac{C_2}{2}||G\mathbf{u}_2 - \mathbf{y}||^2 + \frac{C_4}{2}[||(\mathbf{y} - G\mathbf{u}_2$$
$$- \varepsilon\mathbf{e})_+||^2 - ||(\mathbf{y} - G\mathbf{u}_2 - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+||^2], \tag{13b}$$

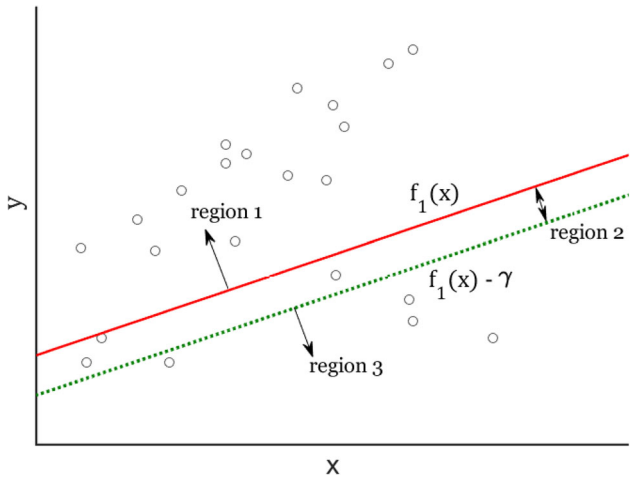

Fig. 3 Illustration of the down-bound function $f_1(x)$ on Huber twin SVR (HTSVR)

where $C_1 > 0, C_2 > 0, C_3 > 0$ and $C_4 > 0$ are input parameters.

Remark 1 When $\varepsilon = 0$, the pair of problems $L_1^{\varepsilon}(.)$ and $L_2^{\varepsilon}(.)$ will become $L_1(.)$ and $L_2(.)$, respectively, i.e., ε-HTSVR formulation becomes the formulation of HTSVR.

Rather than individually deriving the method of solving HTSVR and ε-HTSVR, we describe, as a general study, the method of solving ε-HTSVR. More precisely, we propose to solve the primal problems (12) and (13) by obtaining their critical points using functional iterative and Newton methods.

3.1 Functional iterative method of solving ε-HTSVR (ε-FHTSVR)

Let $\frac{\partial L_1^{\varepsilon}}{\partial \mathbf{u}_1}$ and $\frac{\partial L_2^{\varepsilon}}{\partial \mathbf{u}_2}$ be the gradients of $L_1^{\varepsilon}(\mathbf{u}_1)$ and $L_2^{\varepsilon}(\mathbf{u}_2)$, respectively. Then, since the problem of finding a critical point of (13a) becomes finding a root of $\frac{\partial L_1^{\varepsilon}(\mathbf{u}_1)}{\partial \mathbf{u}_1} = \mathbf{0}$, we solve

$$\left(\frac{I}{C_1} + G^t G\right)\mathbf{u}_1 = G^t\mathbf{y} - \frac{C_3}{C_1}G^t[(G\mathbf{u}_1 - \mathbf{y} - \varepsilon\mathbf{e})_+ - (G\mathbf{u}_1 - \mathbf{y} - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+].$$

This leads to the following simple iterative scheme:

$$\mathbf{u}_1^{i+1} = \left(\frac{I}{C_1} + G^t G\right)^{-1} G^t \left[\mathbf{y} - \frac{C_3}{C_1}[(G\mathbf{u}_1^i - \mathbf{y} - \varepsilon\mathbf{e})_+ - (G\mathbf{u}_1^i - \mathbf{y} - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+]\right]$$
where $i = 0, 1, \ldots$ (14a)

Similarly, $\frac{\partial L_2^{\varepsilon}(\mathbf{u}_2)}{\partial \mathbf{u}_2} = \mathbf{0} \Rightarrow \left(\frac{I}{C_2} + G^t G\right)\mathbf{u}_2 = G^t\mathbf{y} + \frac{C_4}{C_2}G^t[(\mathbf{y} - G\mathbf{u}_2 - \varepsilon\mathbf{e})_+ - (\mathbf{y} - G\mathbf{u}_2 - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+]$ whose solution can be obtained by applying the simple iterative scheme

$$\mathbf{u}_2^{i+1} = \left(\frac{I}{C_2} + G^t G\right)^{-1} G^t \left[\mathbf{y} + \frac{C_4}{C_2}[(\mathbf{y} - G\mathbf{u}_2^i - \varepsilon\mathbf{e})_+ - (\mathbf{y} - G\mathbf{u}_2^i - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+]\right]$$
where $i = 0, 1, \ldots$ (14b)

Combining Eqs. (14a) and (14b), the iterative method of solving problem (13) (ε-FHTSVR) becomes: for $k = 1, 2$ and $i = 0, 1, \ldots$

$$\mathbf{u}_k^{i+1} = \left(\frac{I}{C_k} + G^t G\right)^{-1} G^t \left[\mathbf{y} + (-1)^k \frac{C_{k+2}}{C_k}[((-1)^k(\mathbf{y} - G\mathbf{u}_k^i) - \varepsilon\mathbf{e})_+ - ((-1)^k(\mathbf{y} - G\mathbf{u}_k^i) - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+]\right]. \quad (15)$$

Note that in Mangasarian and Musicant [24], a robust regression method using Huber loss function without the regularization term was considered. However, in our approach the regularization term in 2-norm is employed in the objective function to improve the generalization ability which further makes the objective function strongly convex.

Remark 2 The proposed iterative method requires the computation of the inverse of two matrices of order $(m + 1)$. This suggests that our proposed method is more suitable for problems of medium size.

Remark 3 Assuming $\varepsilon = 0$ in (14) corresponds to the functional iterative method (FHTSVR)

$$\mathbf{u}_k^{i+1} = \left(\frac{I}{C_k} + G^t G\right)^{-1} G^t \left[\mathbf{y} + (-1)^k \frac{C_{k+2}}{C_k}[((-1)^k(\mathbf{y} - G\mathbf{u}_k^i))_+ - ((-1)^k(\mathbf{y} - G\mathbf{u}_k^i) - \gamma\mathbf{e})_+]\right], \quad i = 0, 1, \ldots$$

for solving HTSVR problem given by (12).

3.2 Newton method of solving ε-HTSVR (ε-NHTSVR)

In this subsection, we solve the two unconstrained minimization problems (13a) and (13b) as the solutions of root-finding problems by applying Newton iterative method with Armijo step size [13, 21] and establish the finite termination.

For this purpose, consider the following functions:

$$\mathbf{g}_1(\mathbf{u}_1) = \left(\frac{I}{C_1} + G^t G\right)\mathbf{u}_1 - \left(G^t\mathbf{y} - \frac{C_3}{C_1}G^t[(G\mathbf{u}_1 - \mathbf{y} - \varepsilon\mathbf{e})_+ - (G\mathbf{u}_1 - \mathbf{y} - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+]\right)$$

and

$$\mathbf{g}_2(\mathbf{u}_2) = \left(\frac{I}{C_2} + G^t G\right)\mathbf{u}_2 - \left(G^t\mathbf{y} + \frac{C_4}{C_2}G^t[(\mathbf{y} - G\mathbf{u}_2 - \varepsilon\mathbf{e})_+ - (\mathbf{y} - G\mathbf{u}_2 - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+]\right),$$

or equivalently, we have for $k = 1, 2$,

$$\mathbf{g}_k(\mathbf{u}_k) = \left(\frac{I}{C_k} + G^tG\right)\mathbf{u}_k - \left(G^t\mathbf{y} + (-1)^k \frac{C_{k+2}}{C_k} G^t\Big[((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e})_+ - ((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+\Big]\right) \tag{16}$$

Then, a generalized Hessian [18] of $\mathbf{g}_k(\mathbf{u}_k)$ can be computed to be

$$\partial\mathbf{g}_k(\mathbf{u}_k) = \left(\frac{I}{C_k} + G^tG\right) + \frac{C_{k+2}}{C_k} G^tE_k(\mathbf{u}_k)G, \tag{17}$$

where $E_k(\mathbf{u}_k) = \text{diag}(\text{sign}(((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e})_+)) - \text{diag}(\text{sign}(((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+))$.

Remark 4 It is simple to verify that for any $\mathbf{v}_k \in R^{m+1}$, the matrix $E_k(\mathbf{v}_k)$ is a diagonal matrix such that its diagonal values will be either 0 or 1, which implies the Hessian matrix (17) is always positive definite.

Remark 5 Assuming $\varepsilon = 0$ in (16) and (17) and solving the root-finding problem $\mathbf{g}_k(\mathbf{u}_k) = \mathbf{0}$ by Newton–Armijo algorithm (NHTSVR), the solutions of HTSVR problems (12a) and (12b) will be obtained.

We now describe Newton–Armijo algorithm for solving (13) with $k = 1, 2$.

Start with any initial guess $\mathbf{u}_k^0 \in R^{m+1}$ and let $i = 0$

(i) Stop the iteration if $\mathbf{g}_k(\mathbf{u}_k^i) = \mathbf{0}$

Else

Determine the direction vector $\mathbf{d}_k^i \in R^{m+1}$ as the solution of the following system of linear equations in $m+1$ variables:

$$\partial\mathbf{g}_k(\mathbf{u}_k^i)\mathbf{d}_k^i = -\mathbf{g}_k(\mathbf{u}_k^i)$$

(ii) Armijo stepsize. Define:

$$\mathbf{u}_k^{i+1} = \mathbf{u}_k^i + \lambda_k^i\mathbf{d}_k^i,$$

where the stepsize $\lambda_k^i = \max\{1, \frac{1}{2}, \frac{1}{4}, \ldots\}$ is such that:

$$L_k^\varepsilon(\mathbf{u}_k^i) - L_k^\varepsilon(\mathbf{u}_k^i + \lambda_k^i\mathbf{d}_k^i) \geq -\delta\,\lambda_k^i\mathbf{g}_k(\mathbf{u}_k^i)^t\mathbf{d}_k^i \text{ and } \delta \in (0, \frac{1}{2})$$

(iii) Replace i by $i+1$ and go to (i)

In the next theorem, we establish the global convergence of our algorithm and its finite termination.

Theorem 1 *For the symmetric positive definite matrix defined by (17) and starting from any $\mathbf{u}_k^0 \in R^{m+1}$ where $k = 1, 2$, let $\{\mathbf{u}_k^i\}$ be the sequence of iterates obtained using Newton algorithm with Armijo step size. Then, the sequence $\{\mathbf{u}_k^i\}$ terminates at the global minimum $\mathbf{u}_k \in R^{m+1}$ in a finite number of step sizes.*

Proof Since the objective function of our proposed unconstrained minimization problem (13) is strongly convex, the convergence of the sequence $\{\mathbf{u}_k^i\}$, obtained using Newton algorithm with Armijo step size for solving the root-finding problem of (16), to its global solution $\mathbf{u}_k \in R^{m+1}$ follows from the results of [23, Theorem 2.1, Example 2.1(ii), Example 2.4(iv)].

Now, we prove the finite termination $\{\mathbf{u}_k^i\}$ at $\mathbf{u}_k$ by the simplified Newton method

$$\partial\mathbf{g}_k(\mathbf{u}_k^i)(\mathbf{u}_k^{i+1} - \mathbf{u}_k^i) = -\mathbf{g}_k(\mathbf{u}_k^i) \quad \text{where } i = 0, 1, \ldots \tag{18}$$

Since $\mathbf{u}_k$ is the solution of (12), $\mathbf{g}_k(\mathbf{u}_k) = \mathbf{0}$ implies

$$\left(\frac{I}{C_k} + G^tG\right)\mathbf{u}_k = \left(G^t\mathbf{y} + (-1)^k \frac{C_{k+2}}{C_k} G^t\Big[((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e})_+ - ((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+\Big]\right) \tag{19}$$

Therefore, subtracting (19) from (18), we get

$$\begin{aligned}\left(\frac{I}{C_k} + G^tG\right)(\mathbf{u}_k^{i+1} - \mathbf{u}_k) &= -\frac{C_{k+2}}{C_k} G^tE_k(\mathbf{u}_k^i)G(\mathbf{u}_k^{i+1} - \mathbf{u}_k^i) \\ &+(-1)^k \frac{C_{k+2}}{C_k} G^t \\ &\Big[((-1)^k(\mathbf{y} - G\mathbf{u}_k^i) - \varepsilon\mathbf{e})_+ - ((-1)^k(\mathbf{y} - G\mathbf{u}_k^i) - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+\Big] \\ &- (-1)^k \frac{C_{k+2}}{C_k} G^t\Big[((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e})_+ \\ &- ((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+\Big]\end{aligned} \tag{20}$$

To prove the finite termination, we verify that $\mathbf{u}_k^{i+1} = \mathbf{u}_k$ when $\mathbf{u}_k^i$ is sufficiently close to $\mathbf{u}_k$. Setting $\mathbf{u}_k^{i+1} = \mathbf{u}_k$ in (20), it is sufficient to show that

$$\begin{aligned}&-E_k(\mathbf{u}_k^i)G(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k \\ &\Big[((-1)^k(\mathbf{y} - G\mathbf{u}_k^i) - \varepsilon\mathbf{e})_+ - ((-1)^k(\mathbf{y} - G\mathbf{u}_k^i) - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+\Big] \\ &- (-1)^kG^t\Big[((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e})_+ \\ &-((-1)^k(\mathbf{y} - G\mathbf{u}_k) - \varepsilon\mathbf{e} - \gamma\mathbf{e})_+\Big] = \mathbf{0}\end{aligned} \tag{21}$$

In fact, by taking $r(\mathbf{u}_k) = ((-1)^k(y_\ell - G_\ell\mathbf{u}_k) - \varepsilon)$ for any $\ell = 1, 2, \ldots, m$, where G_ℓ is the ℓth row of the matrix G, we will verify that the ℓth component of (21) will be zero, i.e.

$$\begin{aligned}&\big(\text{sign}(r(\mathbf{u}_k^i)_+) - \text{sign}((r(\mathbf{u}_k^i) - \gamma)_+)\big)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) \\ &+ (-1)^k[r(\mathbf{u}_k)_+ - r(\mathbf{u}_k^i)_+ \\ &+ (r(\mathbf{u}_k^i) - \gamma)_+ - (r(\mathbf{u}_k) - \gamma)_+] = 0.\end{aligned} \tag{22}$$

By considering the following possible cases, we verify (22) when $\mathbf{u}_k^i$ is sufficiently close to $\mathbf{u}_k$.

1. $r(\mathbf{u}_k) > \gamma$,
   - $r(\mathbf{u}_k^i) > \gamma : (1-1)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[r(\mathbf{u}_k) - r(\mathbf{u}_k^i) + (r(\mathbf{u}_k^i) - \gamma) - (r(\mathbf{u}_k) - \gamma)] = 0$
   - $r(\mathbf{u}_k^i) = \gamma$: Not possible when $\mathbf{u}_k^i$ is sufficiently close to $\mathbf{u}_k$
   - $r(\mathbf{u}_k^i) < \gamma$: Not possible when $\mathbf{u}_k^i$ is sufficiently close to $\mathbf{u}_k$
2. $r(\mathbf{u}_k) = \gamma$,
   - $r(\mathbf{u}_k^i) > \gamma : (1-1)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[r(\mathbf{u}_k) - r(\mathbf{u}_k^i) + (r(\mathbf{u}_k^i) - \gamma) - (r(\mathbf{u}_k) - \gamma)] = 0$
   - $r(\mathbf{u}_k^i) = \gamma : (1-0)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[r(\mathbf{u}_k) - r(\mathbf{u}_k^i) + 0 - 0] = G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) = r(\mathbf{u}_k) - r(\mathbf{u}_k^i) = 0$
   - $0 < r(\mathbf{u}_k^i) < \gamma : (1-0)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[r(\mathbf{u}_k) - r(\mathbf{u}_k^i) + 0 - 0]$
     $= G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[((-1)^k(y_\ell - G_\ell\mathbf{u}_k) - \varepsilon) - ((-1)^k(y_\ell - G_\ell\mathbf{u}_k^i) - \varepsilon)] = 0$
3. $0 < r(\mathbf{u}_k) < \gamma$,
   - $r(\mathbf{u}_k^i) > \gamma$: Not possible when $\mathbf{u}_k^i$ is sufficiently close to $\mathbf{u}_k$
   - $r(\mathbf{u}_k^i) = \gamma$: Not possible when $\mathbf{u}_k^i$ is sufficiently close to $\mathbf{u}_k$
   - $r(\mathbf{u}_k^i) < \gamma : (1-0)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[r(\mathbf{u}_k) - r(\mathbf{u}_k^i) + 0 - 0]$
     $= G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[((-1)^k(y_\ell - G_\ell\mathbf{u}_k) - \varepsilon) - ((-1)^k(y_\ell - G_\ell\mathbf{u}_k^i) - \varepsilon)] = 0$
4. $r(\mathbf{u}_k) = 0$,
   - $0 < r(\mathbf{u}_k^i) < \gamma : (1-0)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[0 - r(\mathbf{u}_k^i) + 0 - 0]$
     $= G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[-((-1)^k(y_\ell - G_\ell\mathbf{u}_k^i) - \varepsilon)]$
     $= G_\ell\mathbf{u}_k - y_\ell + (-1)^k\varepsilon = (-1)^{k+1}r(\mathbf{u}_k) = 0$
   - $r(\mathbf{u}_k^i) = 0 : (0-0)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[0-0+0-0] = 0$
   - $r(\mathbf{u}_k^i) < 0 : (0-0)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[0-0+0-0] = 0$
5. $r(\mathbf{u}_k) < 0$,
   - $0 < r(\mathbf{u}_k^i)$: Not possible when $\mathbf{u}_k^i$ is sufficiently close to $\mathbf{u}_k$
   - $r(\mathbf{u}_k^i) = 0$: Not possible when $\mathbf{u}_k^i$ is sufficiently close to $\mathbf{u}_k$
   - $r(\mathbf{u}_k^i) < 0 : (0-0)G_\ell(\mathbf{u}_k - \mathbf{u}_k^i) + (-1)^k[0-0+0-0] = 0$

Remark 6 In the proof of Theorem 1, one can observe that Armijo step size is used only to guarantee the global convergence.

Remark 7 For the numerical study in the next section, experiments were performed using simplified Newton method (18).

4 Experiments and results

To demonstrate the efficacy of the proposed FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR methods in terms of accuracy, learning time and robustness, their performances are compared with SVR, LS-SVR, TSVR and RHSVR on several synthetic and well-known benchmark datasets corrupted with noise and having outliers.

All the experiments were implemented in MATLAB R2008b on a PC with 3.40 GHz Intel® Core™ i7 processor, 8 GB RAM under 64-bit Microsoft Windows 7 operating system. For training SVR, TSVR and RHSVR, the MOSEK optimization toolbox available at http://www.mosek.com was used. However, no external optimizer was used for training LS-SVR, FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR.

Experiments were performed by choosing the popular Gaussian kernel function of the form: $k(\mathbf{x}, \mathbf{z}) = \exp(-\mu||\mathbf{x} - \mathbf{z}||^2)$, where $\mathbf{x}, \mathbf{z} \in R^n$ and $\mu > 0$ is the parameter. Keeping in mind with the increase in computational cost, we set $C_1 = C_2$ and $C_3 = C_4$. In the implementation of FHTSVR and NHTSVR, we assumed $C = C_1 = C_2 = C_3 = C_4$. The termination criteria for the error of tolerance and the maximum iteration were taken as 10^{-2} and 10 respectively. All the parameters were selected by employing the grid search using tenfold cross-validation methodology. The regularization parameters $C, C_1 = C_2$ and $C_3 = C_4$ and the kernel parameter μ were chosen from the set $\{2^i | i = -9, -8, -7, \ldots, 9\}$. However, the parameters $\varepsilon, \varepsilon_1 = \varepsilon_2$ were selected from $\{10^i | i = -3 - 2, -1\}$ and the parameter γ from the set {0.1, 1, 1.345} [24]. By letting T be the number of test samples and $\tilde{y}_i$ be the predicted value for the ith observed value y_i, the root-mean-square error (RMSE): RMSE = $\sqrt{\frac{1}{T}\sum_{i=1}^{T}(y_i - \tilde{y}_i)^2}$ is

Table 1 Types of noise used in experiments. $N(0,\ 0.2^2)$ represents the Gaussian random variable with mean 0 and standard deviation 0.2

Name	Noise distribution
Type A	$N(0,\ 0.2^2)$
Type B	$(1-\rho)N(0,\ 0.2^2) + \rho N(0,\ 2^2), \quad \rho = 0.1$
Type C	$(1-\rho)N(0,\ 0.2^2) + \rho N(0,\ 2^2), \quad \rho = 0.2$
Type D	$(1-\rho)N(0,\ 0.2^2) + \rho N(0,\ 2^2), \quad \rho = 0.3$

Table 2 Performance comparison of our proposed methods FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR with SVR, LS-SVR, TSVR and RHSVR. RMSE was used for comparison. Gaussian kernel was employed. Time is for training in seconds. Bold type shows the best result

Dataset (train size, test size)	Types of noise	SVR (C, μ, ε) Time (Rank)	LS-SVR (C, μ) Time (Rank)	TSVR ($C_1 = C_2$, μ, $\varepsilon_1 = \varepsilon_2$) Time (Rank)	RHSVR (μ, γ) Time (Rank)	FHTSVR (C, μ, γ) Time (Rank)	NHTSVR (C, μ, γ) Time (Rank)	ε-FHTSVR ($C_1 = C_2$, $C_3 = C_4$, μ, $\varepsilon_1 = \varepsilon_2$, γ) Time (Rank)	ε-NHTSVR ($C_1 = C_2$, $C_3 = C_4$, μ, $\varepsilon_1 = \varepsilon_2$, γ) Time (Rank)
Function 1 (300 × 1, 400 × 1)	Type A	0.0481 ± 0.0073 (2^0, 2^0, 10^{-3}) 2.0124 (8)	**0.0393 ± 0.0082** (2^9, 2^{-2}) 0.0940 (1)	0.0464 ± 0.0083 (2^1, 2^{-1}, 10^{-1}) 0.1645 (7)	0.0424 ± 0.0095 (2^{-3}, 1.345) 4.3351 (6)	0.0396 ± 0.0082 (2^5, 2^{-1}, 1) 0.1485 (3)	0.0398 ± 0.0081 (2^5, 2^{-1}, 1) 0.1424 (5)	0.0395 ± 0.0097 (2^{-3}, 2^{-3}, 2^0, 10^{-1}, 1) 0.1349 (2)	0.0397 ± 0.0070 (2^5, 2^4, 2^{-1}, 10^{-3}, 1) 0.1354 (4)
	Type B	0.0527 ± 0.0108 (2^1, 2^{-1}, 10^{-1}) 2.0125 (6)	0.0523 ± 0.0087 (2^4, 2^{-1}) 0.1038 (5)	0.0667 ± 0.0134 (2^3, 2^{-1}, 10^{-3}) 0.1688 (8)	0.0568 ± 0.0105 (2^{-3}, 1) 3.1229 (7)	0.0491 ± 0.0102 (2^3, 2^{-1}, 0.1) 0.1496 (4)	0.0486 ± 0.0109 (2^3, 2^{-1}, 0.1) 0.1456 (2)	**0.0480 ± 0.0126** (2^9, 2^{-3}, 2^0, 10^{-2}, 0.1) 0.1344 (1)	0.0488 ± 0.0091 (2^3, 2^3, 2^{-1}, 10^{-1}, 1) 0.1335 (3)
	Type C	0.0908 ± 0.0213 (2^{-4}, 2^0, 10^{-1}) 2.0114 (8)	0.0812 ± 0.0136 (2^2, 2^{-1}) 0.1037 (5)	0.0907 ± 0.0217 (2^0, 2^{-1}, 10^{-1}) 0.1533 (7)	0.0858 ± 0.0162 (2^{-3}, 1) 2.6414 (6)	**0.0601 ± 0.0162** (2^{-6}, 2^2, 1) 0.1480 (1)	0.0610 ± 0.0199 (2^{-6}, 2^2, 1) 0.1498 (2)	0.0625 ± 0.0144 (2^{-4}, 2^{-2}, 2^0, 10^{-2}, 0.1) 0.1324 (4)	0.0620 ± 0.0157 (2^{-4}, 2^{-3}, 2^0, 10^{-1}, 1) 0.1359 (3)
	Type D	0.1264 ± 0.0237 (2^{-2}, 2^{-1}, 10^{-3}) 2.0146 (7)	0.1204 ± 0.0179 (2^{-2}, 2^{-1}) 0.0971 (5)	0.1302 ± 0.0468 (2^5, 2^{-2}, 10^{-1}) 0.1596 (8)	0.1237 ± 0.0358 (2^{-4}, 0.1) 3.0307 (6)	0.0697 ± 0.0236 (2^{-3}, 2^1, 0.1) 0.1489 (2)	**0.0695 ± 0.0235** (2^1, 2^{-1}, 1.345) 0.1450 (1)	0.0710 ± 0.0280 (2^{-4}, 2^{-4}, 2^0, 10^{-2}, 0.1) 0.1299 (4)	0.0704 ± 0.0235 (2^2, 2^{-5}, 2^0, 10^{-3}, 1) 0.1337 (3)
Function 2 (300 × 1, 400 × 1)	Type A	0.0430 ± 0.0055 (2^1, 2^3, 10^{-1}) 2.0387 (6.5)	**0.0419 ± 0.0033** (2^3, 2^3) 0.9775 (1)	0.0450 ± 0.0066 (2^0, 2^1, 10^{-1}) 0.1614 (8)	0.0421 ± 0.0052 (2^1, 1) 2.9457 (2)	0.0424 ± 0.0045 (2^1, 2^4, 0.1) 0.1599 (4)	0.0430 ± 0.0068 (2^1, 2^4, 0.1) 0.1455 (6.5)	0.0422 ± 0.0074 (2^6, 2^6, 2^3, 10^{-3}, 0.1) 0.1346 (3)	0.0428 ± 0.0060 (2^{-1}, 2^{-1}, 2^5, 10^{-2}, 0.1) 0.1335 (5)
	Type B	0.0664 ± 0.0126 (2^1, 2^4, 10^{-1}) 2.0627 (8)	0.0577 ± 0.0126 (2^3, 2^3) 0.0926 (6)	0.0640 ± 0.0092 (2^{-2}, 2^2, 10^{-3}) 0.1524 (7)	0.0555 ± 0.0145 (2^1, 1) 3.1218 (3.5)	0.0557 ± 0.0134 (2^4, 2^3, 1) 0.1479 (5)	0.0555 ± 0.0132 (2^2, 2^4, 1.345) 0.1489 (3.5)	**0.0552 ± 0.0139** (2^1, 2^{-1}, 2^4, 10^{-1}, 1.345) 0.1352 (1)	0.0554 ± 0.0139 (2^1, 2^{-1}, 2^4, 10^{-1}, 1) 0.1348 (2)

Table 2 (continued)

Dataset (train size, test size)	Types of noise	SVR (C, μ, ε) Time (Rank)	LS-SVR (C, μ) Time (Rank)	TSVR (C_1= C_2, μ, ε_1= ε_2) Time (Rank)	RHSVR (μ, γ) Time (Rank)	FHTSVR (C, μ, γ) Time (Rank)	NHTSVR (C, μ, γ) Time (Rank)	ε-FHTSVR (C_1= C_2, C_3= C_4, μ, ε_1= ε_2, γ) Time (Rank)	ε-NHTSVR (C_1= C_2, C_3= C_4, μ, ε_1= ε_2, γ) Time (Rank)
	Type C	0.0931 ± 0.0131	0.0839 ± 0.0134	0.1040 ± 0.0186	0.0830 ± 0.0185	**0.0802 ± 0.0142**	0.0804 ± 0.0144	0.0804 ± 0.0092	0.0806 ± 0.0166
		(2^7, 2^1, 10^{-3})	(2^9, 2^1)	(2^4, 2^2, 10^{-1})	(2^1, 1)	(2^7, 2^2, 1.345)	(2^7, 2^2, 1.345)	(2^1, 2^4, 2^3, 10^{-1}, 1)	(2^3, 2^2, 2^3, 10^{-1}, 1.345)
		2.0530	0.0990	0.1513	3.1043	0.1446	0.1452	0.1367	0.1352
		(7)	(6)	(8)	(5)	(1)	(2.5)	(2.5)	(4)
	Type D	0.1389 ± 0.0304	0.1118 ± 0.0250	0.1401 ± 0.0268	0.1276 ± 0.0267	0.1053 ± 0.0230	**0.1052 ± 0.0260**	0.1082 ± 0.0223	0.1060 ± 0.0113
		(2^{-2}, 2^4, 10^{-1})	(2^0, 2^4)	(2^3, 2^3, 10^{-3})	(2^2, 1.345)	(2^6, 2^2, 0.1)	(2^5, 2^2, 1.345)	(2^9, 2^9, 2^1, 10^{-1}, 1)	(2^7, 2^4, 2^2, 10^{-2}, 1)
		2.0480	0.1002	0.1509	2.9876	0.1459	0.1142	0.1364	0.1376
		(7)	(5)	(8)	(6)	(2)	(1)	(4)	(3)
Average rank		7.1875	4.25	7.625	5.1875	2.75	2.9375	2.6875	3.375

taken as the measure of accuracy in our study. Further, to get a better picture on the comparative RMSE performance of more algorithms (k) on multiple datasets (N) in terms of statistical tests, we apply, as recommended in Demsar [12], the popular robust nonparametric Friedman test with the corresponding Nemenyi post hoc test:

(a) Under the null hypothesis that all the algorithms are equivalent, the Friedman statistic [12]: $\chi_F^2 = \frac{12N}{k(k+1)}\left(\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right)$ distributed according to χ_F^2 distribution with $(k-1)$ degrees of freedom, where $R_j = \sum_{i=1}^{N} r_j^i/N$ is the average rank of the jth algorithm and r_j^i is the rank of the ith dataset for the jth algorithm, and a better statistic $F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2}$ distributed according to F-distribution with $(k-1, (k-1)(N-1))$ degrees of freedom considered.

(b) If the null hypothesis is rejected, then we proceed with the Nemenyi post hoc test for pairwise comparison of the algorithms by computing the critical difference $\mathrm{CD} = q_\alpha\sqrt{\frac{k(k+1)}{6N}}$, where critical values q_α are based on the studentized range statistic divided by $\sqrt{2}$ [12].

4.1 Synthetic datasets

Firstly, in this subsection, we analyze the performance of the proposed methods in comparison with the other methods on datasets generated using the following two functions studied in Chen et al. [6]:

$$(1)\quad y = \frac{\sin(\pi x)}{\pi x} + \varsigma, \quad x \in [-4, 4];$$

$$(2)\quad y = 15x^4(x^2-1)^2\exp(-x) + \varsigma, \quad x \in [-1, 1],$$

where ς is the additive noise drawn from the noise distributions defined in Table 1.

For each function, 300 training samples are randomly generated and then polluted by adding noise sampled according to noise distribution of Table 1, whereas 400 uniformly spaced samples are selected free of noise for testing. Due to randomness, ten independent trials are performed and their averaged accuracies are tabulated in Table 2. Note that the inputs contaminated with Type B, Type C and Type D contain outliers, but in the case of Type A, only noisy inputs are generated [6]. From the table, one can observe that among all the methods considered, except for Type A noise, the best test accuracy is reported by FHTSVR, NHTSVR or ε-FHTSVR. In the case of Type A noise, LS-SVR shows the best RMSE. Results further demonstrate that in the case of any estimation function

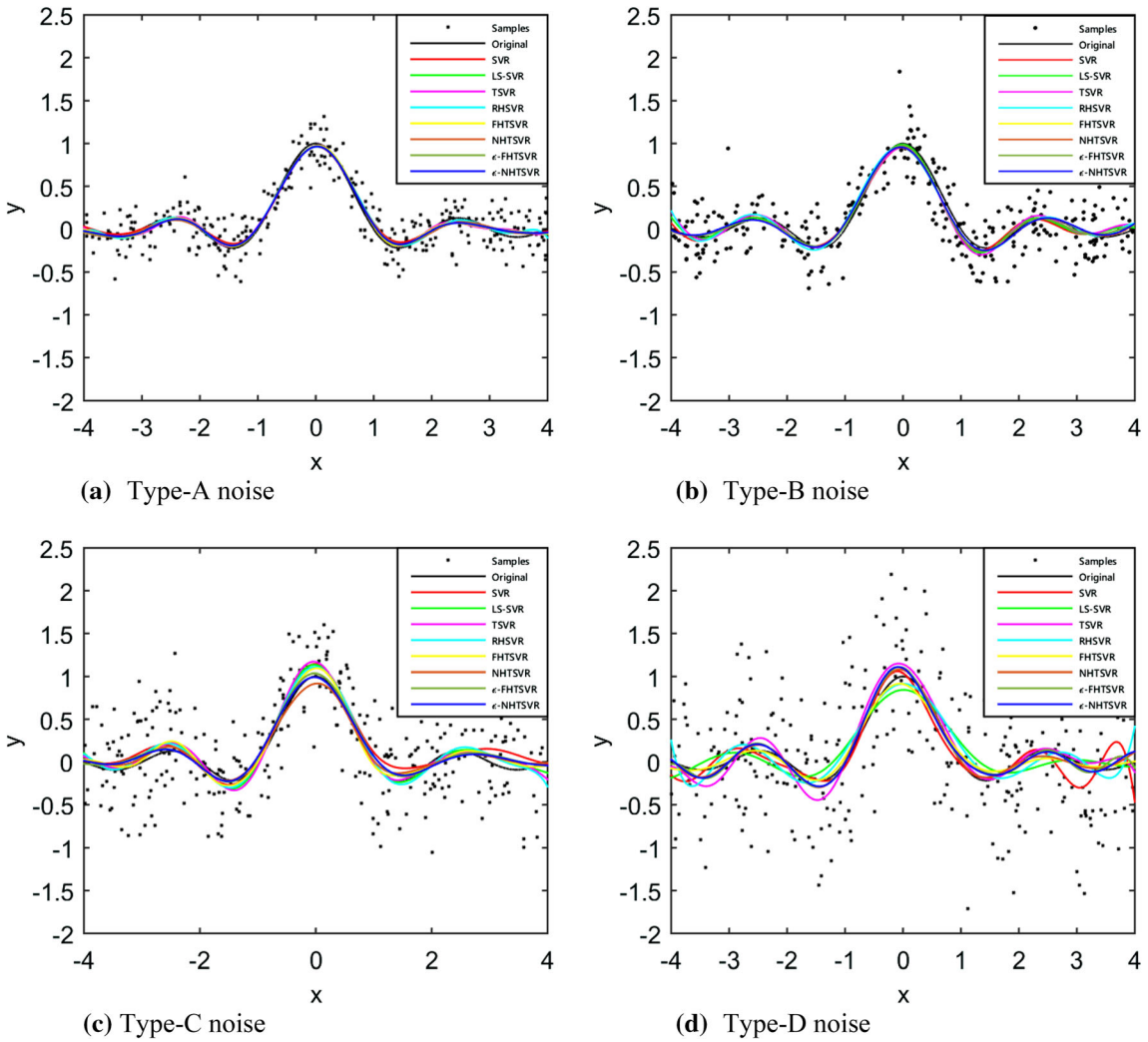

Fig. 4 Accuracy comparison plots for $y = \sin(\pi x)/\pi x$ for inputs corrupted by different types of noise of Table 1

obtained by a learning method considered, as the value for the contamination factor ρ increases, i.e., as we take ρ to be 0%, 10%, 20% and 30%, the prediction accuracy decreases. Comparative accuracy plots for $y = \frac{\sin(\pi x)}{\pi x}$ on test datasets for one sample trial whose inputs are polluted by the noise of Type A to Type D are illustrated in Fig. 4a–d, respectively. Larger deviation of the estimation function plots from the actual curve is clearly visible in Fig. 4c, d, and it is due to the increase in the value of the contamination factor ρ. Comparing Fig. 4a with d, the estimation functions of SVR, LS-SVR, TSVR and RHSVR are much distorted than FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR which is in conformity with the results of RMSE from 0.0481 to 0.1264, 0.0393 to 0.1204, 0.0464 to 0.1302, 0.0424 to 0.1237, 0.0396 to 0.0697, 0.0398 to 0.0695, 0.0395 to 0.0710 and 0.0397 to 0.0704 for SVR, LS-SVR, TSVR, RHSVR, FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR, respectively, i.e., our proposed methods based on Huber functions are more robust than the rest of methods. Similarly, for the synthetic datasets generated by the function $15x^4(x^2-1)^2\exp(-x)$, one-run simulation results are shown in Fig. 5a–d. As in Fig. 4, similar observations can be reported.

For the statistical comparison of the algorithms, we apply Friedman test and Nemenyi post hoc test on the average ranks of Table 2. Clearly, the minimum average rank is shown by ε-FHTSVR and, however, poor prediction accuracy is demonstrated by SVR and TSVR. With $k = 8$ and $N = 8$, now we perform the statistical tests [12]

$$\chi_F^2 = \frac{12\times 8}{8\times 9}\left[7.1875^2 + 4.25^2 + 7.625^2 + 5.1875^2 + 2.75^2 + 2.9375^2 + 2.6875^2 + 3.375^2 - \frac{8\times 9^2}{4}\right] \cong 36.7708$$

$$F_F = \frac{7\times 36.7708}{8\times 7 - 36.7708} \cong 13.3857$$

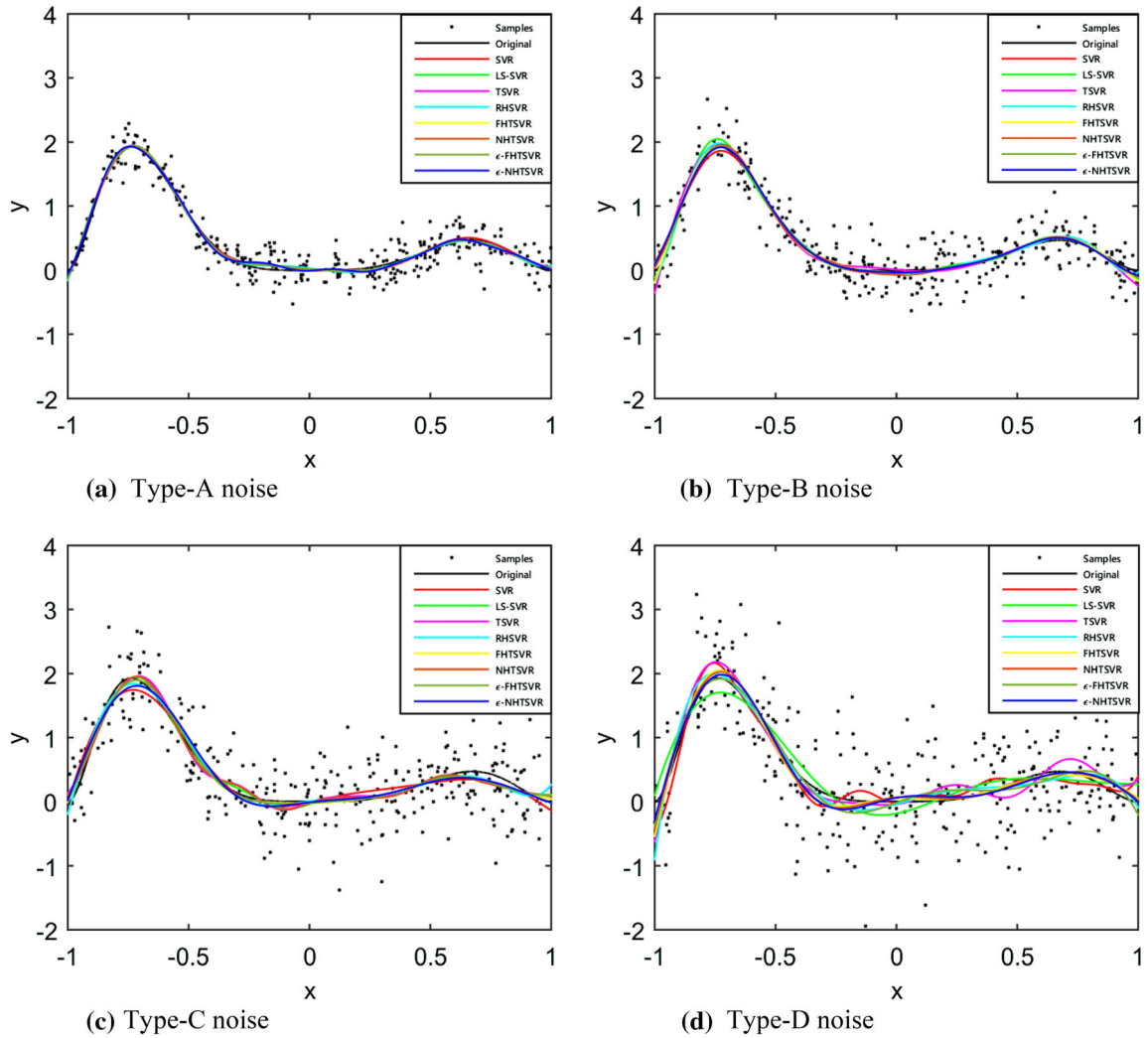

Fig. 5 Accuracy comparison plots for $15x^4(x^2-1)^2\exp(-x)$ for inputs corrupted by different types of noise of Table 1

where F_F is distributed according to F-distribution with $(7, 7\times 7) = (7, 49)$ degree of freedom. Since the F_F value is greater than the critical value of $F(7, 49) = 2.2032$ for the level of significance $\alpha = 0.05$, we reject the null hypothesis. Subsequently, Nemenyi post hoc test is applied for pairwise comparison of algorithms. From Demsar [12], the critical value q_α for $\alpha = 0.10$ is 2.780 and the value of CD is $2.780\sqrt{\frac{8\times 9}{6\times 8}} \cong 3.4048$. In terms of average ranks, the difference between (1) the best and worst of LS-SVR, RHSVR, FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR is: $5.1875 - 2.6875 = 2.5 < 3.4048$, and hence we conclude that the post hoc test is not powerful enough to detect any significant differences between these algorithms; (2) the worst of FHTSVR, NHTSVR, ε-FHTSVR, ε-NHTSVR and the best of SVR, TSVR is: $7.1875 - 3.375 = 3.8125 > 3.4048$, which implies that the performance of FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR is better than SVR and TSVR; and (3) the best and the worst performing algorithms among SVR, LS-SVR, TSVR and RHSVR is: $7.625 - 4.25 = 3.375 < 3.4048$, and hence we conclude that the post hoc test could not detect any significant differences between the algorithms.

The better average ranks by the proposed methods and the above statistical comparisons on datasets having noise/outliers clearly show the superiority of FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR for both the test accuracy and the robustness. Again one can observe from Table 2 that the training time for FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR remains low in comparison with SVR, TSVR and RHSVR, and among all the algorithms, LS-SVR shows the least training time.

To further analyze the robustness and predictive performance of our proposed methods, we generate ten more synthetic datasets using five functions defined in Table 3

Table 3 Functions used for generating synthetic datasets

Name	Function	Domain
Function 1	$\sin(x)\cos(x^2)$	$x \in [0,6]$
Function 2	$\exp(x_1 * \sin(\pi x_2))$	$x_1, x_2 \in [-1,1]$
Function 3	$\frac{40 f_3^1(x)}{f_3^2(x)+f_3^3(x)}$, where $f_3^1(x) = \exp(8((x_1-0.5)^2+(x_2-0.5)^2))$ $f_3^2(x) = \exp(8((x_1-0.2)^2+(x_2-0.7)^2))$ $f_3^3(x) = \exp(8((x_1-0.7)^2+(x_2-0.2)^2))$	$x_1, x_2 \in [0,1]$
Function 4	$1.3356(f_4^1(x_1)+f_4^2(x_2))$ where $f_4^1(x_1) = 1.5(1-x_1)+\exp(2x_1-1)\sin(3\pi(x_1-0.6)^2)$ $f_4^2(x_2) = \exp(3(x_2-0.5))\sin(4\pi(x_2-0.9)^2)$	$x_1, x_2 \in [0,1]$
Function 5	$\exp(2x_1\sin(\pi x_4))+\sin(x_2 x_3)$	$x_1, x_2, x_3, x_4 \in [-0.25, 0.25]$

whose training samples are corrupted by another type of noise and outliers [4, 19]. For Function 1 to Function 4, we generate the training and test samples such that they are evenly spaced along the domain axes. However, for Function 5, they are chosen according to uniform distribution on its domain of definition. In all experiments, 400 training and 800 test samples are generated and among them only the training samples are polluted by adding asymmetric noise and outliers. In this study, asymmetric noise drawn from a Chi-square distribution with mean zero [4, 19] is assumed, i.e., the observed value of a training sample is taken as: $y_i = f(\mathbf{x}_i) + \left(\delta_{\chi^2} - 4\right)$, where δ_{χ^2} follows a Chi-square distribution with 4 degrees of freedom. Further, letting τ_{noise} be the ratio of the variance of the noise $(\delta_{\chi^2} - 4)$ to the variance of $f(\mathbf{x})$, experiments are performed by taking $\tau_{\text{noise}} = 0.05, 0.1$. Besides the noise, outliers are introduced by selecting 5% of the training samples randomly and replacing their observed values drawn from uniform distribution in the range of $f(\mathbf{x})$. By repeating the above procedure ten times, the results of the averaged accuracy and standard deviation are shown in Table 4 along with the averaged training time. Clearly, comparable performance is shown by all the methods. Poorer test accuracy for $\tau_{\text{noise}} = 0.1$ in comparison with $\tau_{\text{noise}} = 0.05$ by all the methods indicates the sensitivity to noise and outliers present in the training data. Among the above ten datasets considered which are corrupted by asymmetric noise and having outliers, FHTSVR shows the best test accuracy four times and is followed by SVR and ε-FHTSVR three times, whereas LS-SVR shows the worst test accuracy nine times. In fact, for most of the datasets, our proposed methods result in the smallest RMSE than SVR, LS-SVR, TSVR and RHSVR.

To get a clear picture on the comparative predictive performance of the eight algorithms, we apply Friedman test and Nemenyi test to their average ranks on the ten datasets reported in Table 4. Under the null hypothesis that all the algorithms are equivalent, we compute $\chi_F^2 = 27.7333$ and $F_F = 5.9054$, where F_F is distributed according to F-distribution with $(7, 63)$ degrees of freedom. However, from the statistical table, the critical value of $F(7, 63)$ for the level of significance $\alpha = 0.05$ is 2.1588 and so we reject the null hypothesis. Since the Friedman test failed and as the minimum average rank is attained by FHTSVR, it is reasonable to suppose its superior accuracy performance in comparison with the remaining algorithms. To verify this, we apply the Nemenyi test for pairwise comparison of the algorithms and report their results. According to Demsar [12], the value of CD is 3.0453. In terms of average ranks, the difference between: (1) the best and the worst among SVR, TSVR, RHSVR, FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR algorithms is: $5.2 - 2.8 = 2.4 < 3.0453$, and hence we conclude, however, that the post hoc test could not detect any significant differences between the algorithms; (2) LS-SVR and the worst of SVR, TSVR, RHSVR, FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR algorithms is: $7.8 - 4.55 = 3.25 > 3.0453$ which implies the performance of LS-SVR is worse than SVR, TSVR, RHSVR, FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR; and (3) LS-SVR and RHSVR algorithms is: $7.8 - 5.2 = 2.6 < 3.0453$, and hence we conclude that there is no significant difference between the two algorithms. Again, we observe from Table 4 that RHSVR results in the maximum time for training. Since LS-SVR solves a system of equations, as expected, it is the fastest method among all the methods. Further, we see that the training timings for FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR are very close to each other and at the same time they are faster than SVR, TSVR and RHSVR.

The improved predictive accuracy performance and comparable training timings with the least training timings shown by LS-SVR reconfirm our assertion that our

Table 4 Performance comparison of our proposed methods FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR with SVR, LS-SVR, TSVR and RHSVR. RMSE was used for comparison. Gaussian kernel was employed. Time is for training in seconds. Bold type shows the best result

Dataset (train size, test size)	Types of noise	SVR (C, μ, ε) Time (Rank)	LS-SVR (C, μ) Time (Rank)	TSVR ($C_1 = C_2$, μ, $\varepsilon_1 = \varepsilon_2$) Time (Rank)	RHSVR (μ, γ) Time (Rank)	FHTSVR (C, μ, γ) Time (Rank)	NHTSVR (C, μ, γ) Time (Rank)	ε-FHTSVR ($C_1 = C_2$, $C_3 = C_4$, μ, $\varepsilon_1 = \varepsilon_2$, γ) Time (Rank)	ε-NHTSVR ($C_1 = C_2$, $C_3 = C_4$, μ, $\varepsilon_1 = \varepsilon_2$, γ) Time (Rank)
Function 1 (400 × 1, 800 × 1)	$r_{noise} = 0.05$	**0.0422 ± 0.0063**	0.0512 ± 0.0093	**0.0422 ± 0.0063**	0.0430 ± 0.008	0.0432 ± 0.0054	0.0438 ± 0.0099	0.0448 ± 0.0100	0.0445 ± 0.0061
		(2^2, 2^3, 10^{-1})	(2^7, 2^3)	(2^{-1}, 2^3, 10^{-1})	(2^1, 0.1)	(2^0, 2^{-1}, 0.1)	(2^7, 2^3, 0.1)	(2^7, 2^7, 2^3, 10^{-3}, 0.1)	(2^9, 2^9, 2^3, 10^{-2}, 0.1)
		3.6571	0.1472	0.4024	3.9263	0.2689	0.2895	0.1639	0.1624
		(1.5)	(8)	(1.5)	(3)	(4)	(5)	(7)	(6)
	$r_{noise} = 0.1$	**0.0545 ± 0.0055**	0.0591 ± 0.0087	0.0548 ± 0.0062	0.0558 ± 0.0084	0.0556 ± 0.0076	0.0560 ± 0.0073	0.0558 ± 0.0098	0.0562 ± 0.0099
		(2^2, 2^3, 10^{-1})	(2^6, 2^3)	(2^{-3}, 2^0, 10^{-2})	(2^2, 0.1)	(2^2, 2^4, 0.1)	(2^7, 2^3, 0.1)	(2^7, 2^9, 2^3, 10^{-3}, 0.1)	(2^6, 2^5, 2^3, 10^{-3}, 0.1)
		3.6354	0.1567	0.3842	3.9289	0.2699	0.2701	0.1648	0.1620
		(1)	(8)	(2)	(4.5)	(3)	(6)	(4.5)	(7)
Function 2 (400 × 2, 800 × 2)	$r_{noise} = 0.05$	0.0619 ± 0.0128	0.0641 ± 0.0167	0.0602 ± 0.0149	0.0620 ± 0.0179	0.0559 ± 0.0083	0.0562 ± 0.0100	**0.0543 ± 0.0081**	0.0556 ± 0.0082
		(2^6, 2^2, 10^{-2})	(2^5, 2^1)	(2^{-1}, 2^{-1}, 10^{-2})	(2^0, 0.1)	(2^9, 2^1, 1)	(2^5, 2^2, 0.1)	(2^9, 2^9, 2^1, 10^{-3}, 0.1)	(2^7, 2^6, 2^3, 10^{-3}, 0.1)
		1.9955	0.1025	0.2976	5.0543	0.1876	0.1678	0.1886	0.1908
		(6)	(8)	(5)	(7)	(3)	(4)	(1)	(2)
	$r_{noise} = 0.1$	0.0708 ± 0.0109	0.0797 ± 0.0133	0.0728 ± 0.0111	0.0756 ± 0.0119	0.0651 ± 0.0078	**0.0649 ± 0.0092**	0.0659 ± 0.0092	0.0660 ± 0.0081
		(2^2, 2^2, 10^{-2})	(2^8, 2^{-4})	(2^7, 2^0, 10^{-2})	(2^{-1}, 1.345)	(2^4, 2^1, 1)	(2^1, 2^2, 0.1)	(2^4, 2^9, 2^1, 10^{-3}, 0.1)	(2^9, 2^5, 2^1, 10^{-2}, 0.1)
		1.9790	0.1010	0.3212	3.9485	0.1877	0.1680	0.1896	0.1920
		(5)	(8)	(6)	(7)	(2)	(1)	(3)	(4)
Function 3 (400 × 2, 800 × 2)	$r_{noise} = 0.05$	0.0363 ± 0.0022	0.0379 ± 0.0057	0.0363 ± 0.0051	0.0367 ± 0.0150	**0.0324 ± 0.0073**	0.0341 ± 0.0085	0.0330 ± 0.0051	0.0337 ± 0.0082
		(2^5, 2^1, 10^{-1})	(2^3, 2^3)	(2^{-1}, 2^{-1}, 10^{-1})	(2^0, 0.1)	(2^4, 2^3, 1.345)	(2^2, 2^2, 0.1)	(2^4, 2^2, 2^3, 10^{-3}, 0.1)	(2^7, 2^0, 2^4, 10^{-3}, 0.1)
		1.9794	0.1031	0.2995	4.4277	0.1863	0.1780	0.1874	0.1890
		(5.5)	(8)	(5.5)	(7)	(1)	(4)	(2)	(3)
	$r_{noise} = 0.1$	0.0399 ± 0.0047	0.0414 ± 0.0091	0.0402 ± 0.0052	0.0408 ± 0.0064	**0.0373 ± 0.0038**	0.0385 ± 0.0064	**0.0373 ± 0.0042**	0.0388 ± 0.0073
		(2^8, 2^1, 10^{-1})	(2^2, 2^3)	(2^{-1}, 2^{-1}, 10^{-1})	(2^0, 0.1)	(2^3, 2^3, 0.1)	(2^4, 2^3, 0.1)	(2^4, 2^4, 2^3, 10^{-3}, 0.1)	(2^3, 2^4, 2^3, 10^{-3}, 0.1)
		1.9858	0.1024	0.2903	5.9845	0.1882	0.1892	0.1866	0.1901
		(5)	(8)	(6)	(7)	(1.5)	(3)	(1.5)	(4)

Table 4 (continued)

Dataset (train size, test size)	Types of noise	SVR (C, μ, ε) Time (Rank)	LS-SVR (C, μ) Time (Rank)	TSVR ($C_1 = C_2$, μ, $\varepsilon_1 = \varepsilon_2$) Time (Rank)	RHSVR (μ, γ) Time (Rank)	FHTSVR (C, μ, γ) Time (Rank)	NHTSVR (C, μ, γ) Time (Rank)	ε-FHTSVR ($C_1 = C_2$, $C_3 = C_4$, μ, $\varepsilon_1 = \varepsilon_2$, γ) Time (Rank)	ε-NHTSVR ($C_1 = C_2$, $C_3 = C_4$, μ, $\varepsilon_1 = \varepsilon_2$, γ) Time (Rank)
Function 4 (400 × 2, 800 × 2)	$r_{noise} = 0.05$	**0.1419 ± 0.0117** (2^2, 2^3, 10^{-1}) 3.3344 (1.5)	0.1436 ± 0.0226 (2^6, 2^3) 0.1768 (6)	0.1428 ± 0.0267 (2^1, 2^4, 10^{-2}) 0.4243 (3)	**0.1419 ± 0.0301** (2^1, 0.1) 5.8248 (1.5)	0.1429 ± 0.0229 (2^6, 2^4, 0.1) 0.3014 (4)	0.1437 ± 0.0188 (2^9, 2^4, 0.1) 0.3008 (7.5)	0.1430 ± 0.0114 (2^6, 2^8, 2^4, 10^{-3}, 0.1) 0.1897 (5)	0.1437 ± 0.0235 (2^9, 2^4, 2^4, 10^{-3}, 0.1) 0.1847 (7.5)
	$r_{noise} = 0.1$	0.1680 ± 0.0181 (2^1, 2^4, 10^{-3}) 3.3531 (3.5)	0.1712 ± 0.0268 (2^1, 2^5) 0.1778 (8)	0.1690 ± 0.0205 (2^{-1}, 2^4, 10^{-3}) 0.4077 (5.5)	0.1690 ± 0.0258 (2^1, 0.1) 5.4664 (5.5)	0.1692 ± 0.0252 (2^4, 2^4, 0.1) 0.2946 (7)	0.1678 ± 0.0180 (2^9, 2^3, 1) 0.2808 (2)	0.1680 ± 0.0383 (2^9, 2^1, 2^4, 10^{-3}, 0.1) 0.1895 (3.5)	**0.1673 ± 0.0230** (2^1, 2^7, 2^4, 10^{-2}, 0.1) 0.1830 (1)
Function 5 (400 × 4, 800 × 4)	$r_{noise} = 0.05$	0.0128 ± 0.0131 (2^6, 2^1, 10^{-2}) 1.9833 (2)	0.0133 ± 0.0089 (2^9, 2^{-1}) 0.1028 (8)	0.0130 ± 0.0012 (2^{-2}, 2^0, 10^{-2}) 0.3074 (4)	0.0131 ± 0.0017 (2^{-4}, 0.1) 3.1188 (6.5)	**0.0125 ± 0.0020** (2^9, 2^0, 0.1) 0.1846 (1)	0.0131 ± 0.0021 (2^9, 2^0, 0.1) 0.1835 (6.5)	0.0130 ± 0.0019 (2^9, 2^7, 2^0, 10^{-3}, 0.1) 0.1886 (4)	0.0130 ± 0.0019 (2^9, 2^9, 2^0, 10^{-3}, 0.1) 0.1891 (4)
	$r_{noise} = 0.1$	0.0141 ± 0.0018 (2^8, 2^{-2}, 10^{-2}) 2.0022 (4.5)	0.0158 ± 0.0036 (2^8, 2^{-1}) 0.1014 (8)	0.0147 ± 0.0024 (2^{-3}, 2^0, 10^{-2}) 0.3065 (7)	0.0140 ± 0.0031 (2^{-3}, 0.1) 3.3991 (3)	**0.0136 ± 0.0017** (2^9, 2^0, 0.1) 0.1874 (1.5)	0.0141 ± 0.0019 (2^8, 2^0, 0.1) 0.1868 (4.5)	**0.0136 ± 0.0023** (2^9, 2^9, 2^0, 10^{-3}, 0.1) 0.1909 (1.5)	0.0145 ± 0.0021 (2^8, 2^8, 2^0, 10^{-3}, 0.1) 0.1930 (6)
Average rank		3.55	7.8	4.55	5.2	2.8	4.35	3.3	4.45

Table 5 Performance comparison of our proposed methods FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR with SVR, LS-SVR, TSVR and RHSVR on real-world datasets. RMSE was used for comparison. Linear kernel was employed. Time is for training in seconds. Bold type shows the best result

Dataset (train size, test size)	SVR (C, ε) Time (Rank)	LS-SVR (C) Time (Rank)	TSVR (C_1= C_2, ε_1= ε_2) Time (Rank)	RHSVR (γ) Time (Rank)	FHTSVR (C_1= C_2, γ) Time (Rank)	NHTSVR (C_1= C_2, γ) Time (Rank)	ε-FHTSVR (C_1= C_2, C_3= C_4, ε_1= ε_2, γ) Time (Rank)	ε-NHTSVR (C_1= C_2, C_3= C_4, ε_1= ε_2, γ) Time (Rank)
Hydraulic actuator (766 × 5, 255 × 5)	0.2790 ± 0.0164 (2^7, 10^{-1}) 3.0439 (7)	0.1078 ± 0.0190 (2^9) 0.0593 (2.5)	**0.1075 ± 0.0116** (2^{-3}, 10^{-1}) 0.9001 (1)	0.3817 ± 0.0208 (1) 0.4861 (8)	0.1078 ± 0.0146 (2^9, 0.1) 0.2420 (2.5)	0.1082 ± 0.0145 (2^9, 0.1) 0.2425 (4)	0.1084 ± 0.0150 (2^9, 2^9, 10^{-3}, 0.1) 0.2184 (5)	0.1086 ± 0.0145 (2^9, 2^9, 10^{-2}, 0.1) 0.2170 (6)
Pyrim (56 × 26, 18 × 26)	**0.1345 ± 0.0534** (2^1, 10^{-2}) 0.0186 (1)	0.1399 ± 0.0628 (2^{-7}) 0.0003 (6)	0.2608 ± 0.0738 (2^{-4}, 10^{-1}) 0.0066 (7)	0.3048 ± 0.1993 (0.1) 0.0064 (8)	0.1350 ± 0.0349 (2^0, 0.1) 0.0009 (2.5)	0.1350 ± 0.0349 (2^0, 0.1) 0.0008 (2.5)	0.1359 ± 0.0336 (2^{-2}, 2^{-2}, 10^{-3}, 0.1) 0.0008 (4)	0.1369 ± 0.0303 (2^{-1}, 2^{-1}, 10^{-3}, 0.1) 0.0008 (5)
Pollution (45 × 15, 15 × 15)	44.4765 ± 10.3421 (2^6, 10^{-3}) 0.0066 (7)	**42.6602 ± 10.1374** (2^2) 0.0002 (1)	43.6701 ± 8.5934 (2^{-9}, 10^{-3}) 0.0060 (6)	80.1342 ± 20.7689 (1.345) 0.0063 (8)	42.7410 ± 8.6108 (2^6, 1.345) 0.0007 (4)	42.8211 ± 8.4902 (2^6, 1.345) 0.0007 (5)	42.6790 ± 6.1080 (2^7, 2^6, 10^{-1}, 0.1) 0.0007 (2)	42.7205 ± 6.9622 (2^7, 2^7, 10^{-3}, 1.345) 0.0006 (3)
Concrete CS (773 × 8, 257 × 8)	10.5092 ± 0.5605 (2^3, 10^{-1}) 3.1509 (5)	10.5622 ± 0.5181 (2^3) 0.0691 (7)	10.5448 ± 0.2233 (2^{-9}, 10^{-3}) 0.5534 (6)	10.8629 ± 0.6927 (1.345) 5.2845 (8)	10.4635 ± 0.5128 (2^9, 0.1) 0.2170 (4)	10.3862 ± 0.4602 (2^{-1}, 0.1) 0.2196 (2)	10.4613 ± 0.4651 (2^{-1}, 2^{-2}, 10^{-3}, 0.1) 0.1976 (3)	**10.1965 ± 0.2745** (2^0, 2^{-1}, 10^{-3}, 0.1) 0.2160 (1)
Sunspots (218 × 5, 72 × 5)	18.2078 ± 1.7027 (2^9, 10^{-3}) 0.1020 (7)	19.3854 ± 1.8539 (2^5) 0.0020 (8)	17.2592 ± 1.4105 (2^7, 10^{-1}) 0.0266 (5)	18.1062 ± 2.0008 (1.345) 0.2541 (6)	16.6674 ± 0.7118 (2^6, 0.1) 0.0069 (3)	16.6942 ± 1.4302 (2^6, 0.1) 0.0065 (4)	**16.3175 ± 1.6288** (2^7, 2^7, 10^{-3}, 1.345) 0.0077 (1)	16.4486 ± 1.7616 (2^9, 2^7, 10^{-2}, 1.345) 0.0071 (2)
Servo (126 × 4, 41 × 4)	1.3556 ± 0.2926 (2^5, 10^{-1}) 0.0282 (7)	1.1721 ± 0.1223 (2^4) 0.0006 (6)	**1.1041 ± 0.1154** (2^1, 10^{-3}) 0.0123 (1)	1.7019 ± 0.3153 (1.345) 0.0281 (8)	1.1403 ± 0.1613 (2^9, 1) 0.0027 (2)	1.1452 ± 0.1453 (2^1, 1.345) 0.0020 (3)	1.1487 ± 0.1724 (2^1, 2^{-1}, 2^0, 10^{-1}, 1) 0.0035 (4)	1.1524 ± 0.1454 (2^{-1}, 2^9, 10^{-2}, 1.345) 0.0030 (5)
Triazines (140 × 58, 46 × 58)	0.2130 ± 0.0141 (2^{-3}, 10^{-2}) 0.0376 (6)	**0.1930 ± 0.0198** (2^{-5}) 0.0016 (1)	0.2237 ± 0.0660 (2^{-2}, 10^{-1}) 0.0281 (7)	0.2357 ± 0.0436 (1) 0.0621 (8)	0.1940 ± 0.0179 (2^5, 2^0, 0.1) 0.0035 (3)	0.1934 ± 0.0179 (2^5, 0.1) 0.0035 (2)	0.1976 ± 0.0278 (2^8, 2^{-2}, 10^{-1}, 0.1) 0.0049 (5)	0.1974 ± 0.0256 (2^{-5}, 2^1, 10^{-2}, 0.1) 0.0042 (4)

Table 5 (continued)

Dataset (train size, test size)	SVR (C, ε) Time (Rank)	LS-SVR (C) Time (Rank)	TSVR (C_1= C_2, ε_1= ε_2) Time (Rank)	RHSVR (γ) Time (Rank)	FHTSVR (C_1= C_2, γ) Time (Rank)	NHTSVR (C_1= C_2, γ) Time (Rank)	ε-FHTSVR (C_1= C_2, C_3= C_4, ε_1= ε_2, γ) Time (Rank)	ε-NHTSVR (C_1= C_2, C_3= C_4, ε_1= ε_2, γ) Time (Rank)
Machine CPU (157 × 7, 52 × 7)	38.3409 ± 20.9327 (2^9, 10^{-1}) 0.0443 (7)	38.8814 ± 19.6838 (2^9) 0.0010 (8)	38.1762 ± 11.0092 (2^6, 10^{-3}) 0.0235 (6)	38.0259 ± 19.6682 (0.1) 0.1064 (5)	**35.2733 ± 10.6333** (2^9, 1.345) 0.0037 (1)	35.2910 ± 10.6129 (2^9, 1.345) 0.0035 (2)	37.2048 ± 13.8919 (2^9, 2^{-2}, 10^{-1}, 1.345) 0.0034 (3)	37.2909 ± 16.4228 (2^9, 2^9, 2^0, 10^{-2}, 1.345) 0.0034 (4)
Wisconsin B.C. (146 × 34, 48 × 34)	5.2688 ± 1.1571 (2^2, 10^{-1}) 0.0674 (7)	5.1279 ± 0.8664 (2^{-1}) 0.0008 (5)	5.1442 ± 0.9509 (2^3, 10^{-3}) 0.0153 (6)	5.2884 ± 0.6591 (0.1) 0.1272 (8)	**4.8626 ± 0.6618** (2^{-2}, 1.345) 0.0032 (1)	4.9698 ± 0.7028 (2^{-2}, 1.345) 0.0034 (2)	4.9895 ± 0.7667 (2^{-2}, 2^{-2}, 10^{-1}, 1.345) 0.0045 (3)	5.0122 ± 1.0117 (2^7, 2^{-5}, 10^{-2}, 1.345) 0.0048 (4)
AutoMPG (294 × 7, 98 × 7)	0.6044 ± 0.0386 (2^5, 10^{-1}) 0.2954 (5)	0.6060 ± 0.0450 (2^4) 0.0045 (7)	0.6049 ± 0.0404 (2^{-2}, 10^{-1}) 0.0536 (6)	0.6083 ± 0.0328 (1.345) 0.9105 (8)	0.5960 ± 0.0390 (2^3, 2^0, 1.345) 0.0166 (3)	0.6042 ± 0.0275 (2^9, 0.1) 0.0178 (4)	**0.5860 ± 0.0387** (2^2, 2^7, 10^{-2}, 1.345) 0.0177 (1)	0.5888 ± 0.0275 (2^2, 2^9, 10^{-2}, 1.345) 0.0184 (2)
Boston (380 × 13, 126 × 13)	4.9053 ± 0.3568 (2^2, 10^{-1}) 0.5746 (2)	4.9583 ± 0.6139 (2^2) 0.0101 (7)	**4.9041 ± 0.3042** (2^0, 10^{-2}) 0.1031 (1)	5.2373 ± 1.0861 (0.1) 1.1564 (8)	4.9168 ± 0.3911 (2^9, 2^0, 1.345) 0.0348 (5)	4.9083 ± 0.3742 (2^9, 1.345) 0.0342 (3)	4.9254 ± 0.4726 (2^9, 2^0, 10^{-1}, 1.345) 0.0344 (6)	4.9127 ± 0.5284 (2^1, 2^9, 10^{-2}, 1) 0.0340 (4)
Bodyfat (189 × 14, 63 × 14)	0.0504 ± 0.0315 (2^3, 10^{-3}) 0.0843 (8)	**0.0471 ± 0.0098** (2^9) 0.0016 (1)	0.0488 ± 0.0111 (2^{-2}, 10^{-1}) 0.0356 (6)	0.0493 ± 0.0200 (0.1) 0.1766 (7)	0.0473 ± 0.0173 (2^4, 2^0, 0.1) 0.0065 (2.5)	0.0473 ± 0.0172 (2^4, 0.1) 0.0062 (2.5)	0.0476 ± 0.0182 (2^3, 2^4, 10^{-3}, 0.1) 0.0054 (5)	0.0475 ± 0.0123 (2^1, 2^3, 10^{-2}, 0.1) 0.0051 (4)
NO2 (375 × 7, 125 × 7)	0.5410 ± 0.0288 (2^0, 10^{-2}) 0.3520 (5.5)	0.5412 ± 0.0328 (2^3) 0.0105 (7)	0.5410 ± 0.0258 (2^{-2}, 10^{-3}) 0.0918 (5.5)	0.5706 ± 0.1980 (1) 2.1367 (8)	0.5359 ± 0.0253 (2^7, 2^0, 1) 0.0323 (3)	0.5395 ± 0.0330 (2^{-1}, 1) 0.0332 (4)	**0.5294 ± 0.0275** (2^1, 2^5, 10^{-1}, 0.1) 0.0296 (1)	0.5312 ± 0.0460 (2^3, 2^9, 10^{-2}, 1) 0.0298 (2)
Forest fires (388 × 12, 129 × 12)	57.8416 ± 37.1283 (2^7, 10^{-1}) 0.6512 (6)	59.0618 ± 31.5549 (2^{-8}) 0.0116 (8)	58.4754 ± 32.5352 (2^6, 10^{-3}) 0.1264 (7)	53.8699 ± 35.8501 (1.345) 1.0342 (3)	54.7589 ± 34.145 (2^{-9}, 2^0, 1.345) 0.0332 (5)	**50.3115 ± 34.8697** (2^{-9}, 1.345) 0.0330 (1)	54.3201 ± 37.5285 (2^{-2}, 2^{-2}, 10^{-3}, 1.345) 0.0294 (4)	53.450 ± 38.5145 (2^{-9}, 2^{-9}, 10^{-3}, 1.345) 0.0291 (2)

Table 5 (continued)

Dataset (train size, test size)	SVR (C, ε) Time (Rank)	LS-SVR (C) Time (Rank)	TSVR (C_1= C_2, ε_1= ε_2) Time (Rank)	RHSVR (γ) Time (Rank)	FHTSVR (C_1= C_2, γ) Time (Rank)	NHTSVR (C_1= C_2, γ) Time (Rank)	ε-FHTSVR (C_1= C_2, C_3= C_4, ε_1= ε_2, γ) Time (Rank)	ε-NHTSVR (C_1= C_2, C_3= C_4, ε_1= ε_2, γ) Time (Rank)
Citigroup (563 × 5, 187 × 5)	0.8130 ± 0.0676 (2^8, 10^{-1}) 0.9262 (5)	0.8140 ± 0.0631 (2^8) 0.0316 (6)	**0.8119 ± 0.0365** (2^1, 10^{-3}) 0.2731 (1)	0.9816 ± 0.1116 (1) 7.3542 (8)	0.8121 ± 0.0679 (2^6, 1.345) 0.0912 (2)	0.8124 ± 0.0792 (2^9, 1.345) 0.0922 (3)	0.8129 ± 0.0459 (2^9, 2^9, 10^{-3}, 1.345) 0.0896 (4)	0.8156 ± 0.0470 (2^9, 2^9, 10^{-2}, 1.345) 0.0904 (7)
Google (563 × 5, 187 × 5)	11.7899 ± 0.9954 (2^3, 10^{-1}) 0.8836 (6)	11.9404 ± 1.1648 (2^7) 0.0323 (7)	11.6925 ± 0.9143 (2^3, 10^{-1}) 0.2384 (5)	11.9980 ± 1.9988 (1.345) 2.5604 (8)	11.3507 ± 1.4999 (2^9, 1.345) 0.0912 (4)	11.3502 ± 0.9386 (2^9, 1.345) 0.0910 (3)	11.3284 ± 1.3349 (2^9, 2^9, 10^{-1}, 1.345) 0.0834 (2)	**11.3281 ± 1.27551** (2^9, 2^9, 10^{-3}, 1.345) 0.0829 (1)
IBM (563 × 5, 187 × 5)	1.5898 ± 0.1534 (2^9, 10^{-3}) 1.5348 (5)	1.6031 ± 0.1353 (2^6) 0.0275 (8)	1.5970 ± 0.1662 (2^3, 10^{-1}) 0.2638 (6)	1.6001 ± 0.1239 (1.345) 2.1233 (7)	**1.5460 ± 0.0903** (2^6, 1.345) 0.0960 (1)	1.5487 ± 0.0878 (2^9, 1.345) 0.0956 (2)	1.5508 ± 0.1127 (2^7, 2^7, 10^{-3}, 1.345) 0.0842 (3)	1.5678 ± 0.1714 (2^9, 2^{-1}, 10^{-3}, 1.345) 0.0838 (4)
SNP500 (563 × 5, 187 × 5)	18.766 ± 2.1256 (2^9, 10^{-1}) 1.0801 (3)	19.0827 ± 1.8099 (2^6) 0.0276 (8)	18.7920 ± 1.947 (2^6, 10^{-3}) 0.2234 (5)	18.9870 ± 1.6346 (0.1) 2.3037 (7)	18.7961 ± 1.2234 (2^5, 1.345) 0.0924 (6)	18.7663 ± 1.4065 (2^9, 1.345) 0.0919 (4)	18.6788 ± 1.0472 (2^9, 2^9, 10^{-1}, 1.345) 0.0825 (2)	**18.6663 ± 1.8840** (2^9, 2^9, 10^{-2}, 1.345) 0.0822 (1)
Intel (563 × 5, 187 × 5)	0.4770 ± 0.0507 (2^9, 10^{-1}) 1.3436 (6)	0.4910 ± 0.0349 (2^6) 0.0281 (7)	**0.4719 ± 0.0411** (2^0, 10^{-1}) 0.2503 (1)	0.4965 ± 0.0568 (0.1) 2.4481 (8)	0.4741 ± 0.0336 (2^6, 1.345) 0.0983 (2)	0.4743 ± 0.0420 (2^8, 1) 0.0980 (3)	0.4760 ± 0.0384 (2^9, 2^9, 10^{-1}, 0.1) 0.0882 (4.5)	0.4760 ± 0.0380 (2^9, 2^9, 10^{-2}, 1) 0.0875 (4.5)
Microsoft (563 × 5, 187 × 5)	0.5456 ± 0.0596 (2^9, 10^{-3}) 1.3468 (6)	0.5465 ± 0.0497 (2^9) 0.0285 (7)	**0.5405 ± 0.0548** (2^0, 10^{-1}) 0.2534 (1)	0.5666 ± 0.0492 (0.1) 2.5687 (8)	0.5415 ± 0.0368 (2^7, 1) 0.0949 (4)	0.5448 ± 0.0552 (2^8, 0.1) 0.0946 (5)	0.5408 ± 0.0498 (2^9, 2^9, 10^{-3}, 0.1) 0.0792 (2)	0.5410 ± 0.0565 (2^9, 2^7, 10^{-3}, 1) 0.0784 (3)
Redhat (563 × 5, 187 × 5)	0.6589 ± 0.1670 (2^9, 10^{-3}) 1.3493 (5)	0.6790 ± 0.1789 (2^9) 0.0315 (8)	0.6590 ± 0.1798 (2^{-1}, 10^{-1}) 0.2875 (6)	0.6599 ± 0.1101 (1.345) 3.0771 (7)	0.6470 ± 0.0879 (2^9, 1) 0.0977 (2)	0.6494 ± 0.1188 (2^9, 1) 0.0971 (4)	**0.6456 ± 0.0780** (2^9, 2^9, 10^{-3}, 1) 0.0894 (1)	0.6478 ± 0.0824 (2^9, 2^9, 10^{-3}, 1) 0.0890 (3)
Average Rank	5.5476	5.9762	4.5476	7.3333	2.9762	3.0952	3.1190	3.4048

Table 6 Performance comparison of our proposed methods FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR with SVR, LS-SVR, TSVR and RHSVR on benchmark datasets. RMSE was used for comparison. Gaussian kernel was employed. Time is for training in seconds. Bold type shows the best result

Dataset (train size, test Size)	SVR (C, μ, ε) Time (Rank)	LS-SVR (C, μ) Time (Rank)	TSVR (C_1= C_2, μ, ε_1= ε_2) Time (Rank)	RHSVR (μ, γ) Time (Rank)	FHTSVR (C, μ, γ) Time (Rank)	NHTSVR (C, μ, γ) Time (Rank)	ε-FHTSVR (C_1= C_2, C_3= C_4, μ, ε_1= ε_2, γ) Time (Rank)	ε-NHTSVR (C_1= C_2, C_3= C_4, μ, ε_1= ε_2, γ) Time (Rank)
Hydraulic actuator (766 × 5, 255 × 5)	**0.1050 ± 0.0145** (2^2, 2^3, 10^{-2}) 13.5484 (1.5)	0.1074 ± 0.0144 (2^7, 2^{-1}) 0.6879 (8)	0.1061 ± 0.0182 (2^{-2}, 2^{-2}, 10^{-3}) 1.4506 (5)	**0.1050 ± 0.0194** (2^{-3}, 0.1) 30.9326 (1.5)	0.1070 ± 0.0134 (2^9, 2^{-2}, 0.1) 1.1175 (6)	0.1060 ± 0.0146 (2^9, 2^{-1}, 0.1) 1.0112 (4)	0.1058 ± 0.0110 (2^9, 2^9, 2^{-1}, 10^{-3}, 0.1) 1.0576 (3)	0.1072 ± 0.0135 (2^9, 2^9, 2^0, 10^{-3}, 0.1) 1.1050 (7)
Pyrim (56 × 26, 18 × 26)	0.0969 ± 0.0192 (2^0, 2^{-4}, 10^{-1}) 0.0423 (2)	0.1035 ± 0.0433 (2^1, 2^{-9}) 0.0025 (7)	0.0997 ± 0.0413 (2^{-2}, 2^{-9}, 10^{-3}) 0.0148 (5)	0.2585 ± 0.0774 (2^0, 1.345) 0.0126 (8)	**0.0842 ± 0.0308** (2^0, 2^{-3}, 1.345) 0.0123 (1)	0.0989 ± 0.0424 (2^3, 2^{-6}, 0.1) 0.0085 (3)	0.1006 ± 0.0381 (2^9, 2^{-2}, 2^{-4}, 10^{-2}, 0.1) 0.0073 (6)	0.0996 ± 0.0277 (2^{-3}, 2^2, 2^{-3}, 10^{-3}, 0.1) 0.0057 (4)
Pollution (45 × 15, 15 × 15)	40.4733 ± 8.6221 (2^9, 2^{-4}, 10^{-3}) 0.0257 (6)	41.843 ± 10.0432 (2^6, 2^{-2}) 0.0016 (7)	39.0206 ± 9.2928 (2^7, 2^{-5}, 10^{-1}) 0.0072 (5)	66.1342 ± 9.3633 (2^{-3}, 1) 0.0073 (8)	37.9734 ± 9.5874 (2^{-3}, 2^{-3}, 0.1) 0.0065 (3)	38.6264 ± 12.1540 (2^9, 2^{-4}, 1.345) 0.0070 (4)	**35.5765 ± 8.8987** (2^7, 2^7, 2^{-4}, 10^{-3}, 1.345) 0.0049 (1)	36.4591 ± 6.5890 (2^9, 2^9, 2^{-4}, 10^{-3}, 1.345) 0.0034 (2)
Concrete CS (773 × 8, 257 × 8)	6.0947 ± 0.6432 (2^9, 2^1, 10^{-1}) 13.7346 (2)	**5.9526 ± 0.5917** (2^6, 2^2) 0.7075 (1)	6.1158 ± 0.4132 (2^5, 2^1, 10^{-1}) 1.4933 (3)	6.2653 ± 0.5297 (2^{-3}, 1.345) 31.7790 (7)	6.2780 ± 0.5100 (2^9, 2^1, 1.345) 1.1817 (8)	6.1781 ± 0.4401 (2^9, 2^1, 1.345) 1.0365 (6)	6.1243 ± 0.4323 (2^9, 2^9, 2^1, 10^{-3}, 1.345) 1.1113 (5)	6.1207 ± 0.4980 (2^9, 2^9, 2^1, 10^{-3}, 1.345) 1.2620 (4)
Sunspots (218 × 5, 72 × 5)	13.6197 ± 1.2296 (2^9, 2^1, 10^{-1}) 0.5625 (5)	13.8471 ± 1.1947 (2^6, 2^0) 0.0314 (7)	13.8326 ± 1.3685 (2^2, 2^0, 10^{-1}) 0.0617 (6)	14.2173 ± 1.5332 (2^{-7}, 1) 0.7123 (8)	**13.0556 ± 1.4429** (2^9, 2^0, 1) 0.0618 (1)	13.1496 ± 1.2960 (2^9, 2^1, 1.345) 0.0621 (2)	13.418 ± 0.9022 (2^9, 2^8, 2^1, 10^{-3}, 1.345) 0.0830 (4)	13.2316 ± 1.2960 (2^9, 2^9, 2^1, 10^{-3}, 1.345) 0.0802 (3)
Servo (126 × 4, 41 × 4)	0.6548 ± 0.2094 (2^5, 2^0, 10^{-1}) 0.1852 (5)	0.6692 ± 0.1175 (2^6, 2^3) 0.0104 (7)	0.6685 ± 0.0771 (2^1, 2^0, 10^{-1}) 0.0216 (6)	0.7319 ± 0.2341 (2^{-7}, 1) 0.1733 (8)	0.6378 ± 0.1154 (2^9, 2^3, 1) 0.0203 (4)	0.6295 ± 0.1415 (2^3, 2^1, 1) 0.0397 (3)	0.6130 ± 0.1140 (2^1, 2^9, 2^2, 10^{-3}, 1) 0.0310 (2)	**0.6127 ± 0.0903** (2^9, 2^1, 2^1, 10^{-3}, 1) 0.0340 (1)
Triazines (140 × 58, 46 × 58)	0.1553 ± 0.0214 (2^{-1}, 2^{-3}, 10^{-3}) 0.4067 (5)	**0.1538 ± 0.3012** (2^2, 2^{-9}) 0.0363 (1)	0.1551 ± 0.0182 (2^{-4}, 2^{-9}, 10^{-3}) 0.0469 (4)	0.2191 ± 0.2134 (2^0, 1.345) 0.0657 (8)	0.1555 ± 0.0292 (2^{-1}, 2^{-5}, 0.1) 0.0406 (6)	0.1585 ± 0.0166 (2^1, 2^{-4}, 0.1) 0.0408 (7)	0.1540 ± 0.0278 (2^7, 2^{-2}, 2^{-5}, 10^{-3}, 0.1) 0.0379 (2)	0.1546 ± 0.0231 (2^1, 2^{-2}, 2^1, 10^{-3}, 0.1) 0.0445 (3)

Table 6 (continued)

Dataset (train size, test Size)	SVR (C, μ, ε) Time (Rank)	LS-SVR (C, μ) Time (Rank)	TSVR (C_1= C_2, μ, ε_1= ε_2) Time (Rank)	RHSVR (μ, γ) Time (Rank)	FHTSVR (C, μ, γ) Time (Rank)	NHTSVR (C, μ, γ) Time (Rank)	ε-FHTSVR (C_1= C_2, C_3= C_4, μ, ε_1= ε_2, γ) Time (Rank)	ε-NHTSVR (C_1= C_2, C_3= C_4, μ, ε_1= ε_2, γ) Time (Rank)
Machine CPU (157 × 7, 52 × 7)	27.7046 ± 15.3155 (2^9, 2^{-3}, 10^{-1}) 0.2909 (8)	24.3153 ± 20.0136 (2^9, 2^{-4}) 0.0171 (6)	23.5842 ± 19.3036 (2^7, 2^{-2}, 10^{-1}) 0.0391 (4)	25.752 ± 23.7275 (2^{-3}, 1.345) 0.3330 (7)	22.7457 ± 7.841 (2^9, 2^{-3}, 1.345) 0.0272 (2)	**20.5281 ± 6.9365** (2^9, 2^{-3}, 1.345) 0.0268 (1)	24.2507 ± 14.6245 (2^9, 2^9, 2^{-2}, 10^{-3}, 1.345) 0.0433 (5)	23.3745 ± 14.5017 (2^9, 2^9, 2^{-2}, 10^{-3}, 1.345) 0.0490 (3)
Wisconsin B.C.(146 × 34, 48 × 34)	4.9263 ± 1.1301 (2^9, 2^{-7}, 10^{-1}) 0.2515 (6)	4.9547 ± 0.6647 (2^9, 2^{-9}) 0.0178 (7)	4.8858 ± 0.9596 (2^2, 2^{-8}, 10^{-3}) 0.0367 (4)	5.2523 ± 0.6591 (2^2, 0.1) 0.0354 (8)	**4.5557 ± 0.6716** (2^3, 2^{-4}, 1) 0.0314 (1)	4.8661 ± 0.9126 (2^8, 2^{-7}, 1.345) 0.0315 (3)	4.8480 ± 0.8363 (2^{-2}, 2^9, 2^{-5}, 10^{-3}, 1.345) 0.0396 (2)	4.8950 ± 1.0156 (2^1, 2^4, 2^{-3}, 10^{-3}, 1.345) 0.0321 (5)
AutoMPG (294 × 7, 98 × 7)	0.5820 ± 0.0680 (2^8, 2^{-3}, 10^{-3}) 1.0428 (7)	0.5748 ± 0.0527 (2^2, 2^2) 0.0681 (5)	0.5687 ± 0.0422 (2^{-2}, 2^0, 10^{-1}) 0.1366 (4)	0.5873 ± 0.0404 (2^{-6}, 0.1) 1.6683 (8)	0.5664 ± 0.0446 (2^9, 2^0, 1) 0.1138 (3)	**0.5622 ± 0.0235** (2^9, 2^1, 0.1) 0.1128 (1)	0.5788 ± 0.0323 (2^3, 2^9, 2^0, 10^{-3}, 1.345) 0.1502 (6)	0.5663 ± 0.0409 (2^9, 2^0, 2^2, 10^{-3}, 1.345) 0.1508 (2)
Boston (380 × 13, 126 × 13)	3.4874 ± 0.6286 (2^9, 2^3, 10^{-1}) 1.7704 (8)	3.2138 ± 0.5030 (2^8, 2^{-1}) 0.1252 (4)	3.2126 ± 0.4333 (2^2, 2^{-1}, 10^{-1}) 0.2269 (3)	3.4217 ± 0.5627 (2^{-8}, 0.1) 3.5421 (7)	**3.1088 ± 0.5277** (2^9, 2^0, 0.1) 0.1740 (1)	3.2849 ± 0.5112 (2^9, 2^0, 1.345) 0.1534 (5)	3.1443 ± 0.4926 (2^9, 2^9, 2^0, 10^{-1}, 1.345) 0.2575 (2)	3.2989 ± 0.5688 (2^9, 2^9, 2^0, 10^{-3}, 1) 0.2754 (6)
Bodyfat (189 × 14, 63 × 14)	**0.0173 ± 0.0018** (2^{-4}, 2^{-9}, 10^{-3}) 0.7356 (1)	0.0193 ± 0.0076 (2^{-3}, 2^{-4}) 0.0527 (3)	0.0190 ± 0.0070 (2^{-3}, 2^{-9}, 10^{-1}) 0.0849 (2)	0.0196 ± 0.0195 (2^{-9}, 0.1) 0.5679 (5)	0.0195 ± 0.0053 (2^7, 2^{-9}, 0.1) 0.0670 (4)	0.0198 ± 0.0043 (2^{-7}, 2^{-9}, 0.1) 0.0580 (6)	0.0205 ± 0.0039 (2^{-2}, 2^{-2}, 2^{-5}, 10^{-3}, 0.1) 0.0661 (7)	0.0210 ± 0.0052 (2^{-4}, 2^8, 2^{-5}, 10^{-3}, 0.1) 0.0636 (8)
NO2 (375 × 7, 125 × 7)	0.5213 ± 0.0216 (2^3, 2^0, 10^{-1}) 1.7032 (5)	0.5280 ± 0.0339 (2^2, 2^1) 0.1093 (7)	0.5221 ± 0.0196 (2^0, 2^{-2}, 10^{-1}) 0.2155 (6)	0.5691 ± 0.1894 (2^{-7}, 1) 2.9038 (8)	0.5042 ± 0.0345 (2^2, 2^0, 1) 0.1739 (2)	0.5083 ± 0.0298 (2^5, 2^1, 1) 0.1256 (3)	**0.5006 ± 0.0286** (2^2, 2^9, 2^{-2}, 10^{-1}, 1) 0.2405 (1)	0.5142 ± 0.0485 (2^9, 2^0, 2^1, 10^{-3}, 1) 0.2504 (4)
Forest fires (388 × 12, 129 × 12)	55.7507 ± 32.9831 (2^9, 2^{-1}, 10^{-3}) 1.8952 (6)	56.7026 ± 29.9275 (2^0, 2^{-9}, 10^{-2}) 0.1122 (8)	56.3195 ± 36.3843 (2^5, 2^{-9}, 10^{-3}) 0.2524 (7)	54.9239 ± 35.4559 (2^{-7}, 1.345) 3.9925 (5)	54.3827 ± 32.5311 (2^{-5}, 2^9, 1.345) 0.2626 (4)	**48.4821 ± 34.7688** (2^{-8}, 2^9, 0.1) 0.1568 (1)	51.6733 ± 35.9719 (2^{-2}, 2^{-2}, 2^{-5}, 10^{-3}, 1.345) 0.2570 (3)	50.3803 ± 30.0159 (2^{-8}, 2^{-7}, 2^9, 10^{-3}, 0.1) 0.3404 (2)

Table 6 (continued)

Dataset (train size, test Size)	SVR (C, μ, ε) Time (Rank)	LS-SVR (C, μ) Time (Rank)	TSVR (C_1= C_2, μ, ε_1= ε_2) Time (Rank)	RHSVR (μ, γ) Time (Rank)	FHTSVR (C, μ, γ) Time (Rank)	NHTSVR (C, μ, γ) Time (Rank)	ε-FHTSVR (C_1= C_2, C_3= C_4, μ, ε_1= ε_2, γ) Time (Rank)	ε-NHTSVR (C_1= C_2, C_3= C_4, μ, ε_1= ε_2, γ) Time (Rank)
Citigroup (563 × 5, 187 × 5)	0.7897 ± 0.0676 (2^8, 2^{-1}, 10^{-1}) 7.0814 (5)	0.7928 ± 0.0460 (2^9, 2^{-3}) 0.3574 (8)	**0.7881 ± 0.0501** (2^2, 2^{-1}, 10^{-3}) 0.6982 (1)	0.7887 ± 0.0495 (2^{-9}, 1.345) 21.9228 (3)	0.7884 ± 0.0688 (2^9, 2^2, 1) 0.5790 (2)	0.7910 ± 0.0414 (2^9, 2^0, 1.345) 0.5120 (6)	0.7891 ± 0.0476 (2^9, 2^7, 2^2, 10^{-3}, 1.345) 0.5592 (4)	0.7926 ± 0.0402 (2^9, 2^9, 2^2, 10^{-3}, 1) 0.5882 (7)
Google (563 × 5, 187 × 5)	11.442 ± 0.9954 (2^9, 2^{-1}, 10^{-1}) 4.0585 (6)	11.3203 ± 0.5849 (2^9, 2^{-5}) 0.2163 (4)	11.5839 ± 1.2243 (2^2, 2^1, 10^{-1}) 0.4690 (7)	11.6622 ± 0.7988 (2^{-9}, 1.345) 8.4259 (8)	11.2908 ± 1.0421 (2^9, 2^{-1}, 1.345) 0.3609 (3)	11.3642 ± 0.9278 (2^9, 2^{-1}, 1.345) 0.3618 (5)	**11.2222 ± 0.8495** (2^9, 2^9, 2^{-1}, 10^{-1}, 1.345) 0.5422 (1)	11.2877 ± 0.9472 (2^9, 2^9, 2^{-1}, 10^{-3}, 1.345) 0.5312 (2)
IBM (563 × 5, 187 × 5)	1.5790 ± 0.0894 (2^9, 2^{-3}, 10^{-3}) 4.0464 (6)	1.6009 ± 0.1266 (2^7, 2^{-2}) 0.2155 (8)	1.5713 ± 0.1259 (2^2, 2^{-2}, 10^{-1}) 0.4777 (5)	1.5831 ± 0.1139 (2^{-9}, 1.345) 7.0155 (7)	**1.5315 ± 0.0741** (2^9, 2^{-1}, 1) 0.3737 (1)	1.5388 ± 0.1470 (2^9, 2^{-2}, 1.345) 0.2978 (2)	1.5487 ± 0.0767 (2^9, 2^9, 2^1, 10^{-3}, 1) 0.5574 (4)	1.5463 ± 0.1214 (2^9, 2^9, 2^{-2}, 10^{-3}, 1.345) 0.5719 (3)
SNP500 (563 × 5, 187 × 5)	18.1197 ± 2.1130 (2^9, 2^0, 10^{-1}) 6.9692 (8)	17.6611 ± 1.8781 (2^9, 2^{-5}) 0.3552 (2)	17.983 ± 1.4149 (2^5, 2^2, 10^{-1}) 0.6923 (7)	**17.4243 ± 1.5501** (2^{-4}, 0.1) 10.7769 (1)	17.8479 ± 1.7968 (2^9, 2^{-1}, 1.345) 0.5596 (4)	17.8784 ± 1.2479 (2^9, 2^1, 1.345) 0.5598 (6)	17.7012 ± 1.3409 (2^9, 2^9, 2^{-1}, 10^{-1}, 1.345) 0.5437 (3)	17.8488 ± 1.214 (2^9, 2^9, 2^{-1}, 10^{-3}, 1.345) 0.5366 (5)
Intel (563 × 5, 187 × 5)	0.4663 ± 0.0507 (2^9, 2^{-4}, 10^{-1}) 4.0385 (5)	0.4711 ± 0.0252 (2^7, 2^{-1}) 0.2145 (7)	0.4642 ± 0.0503 (2^{-1}, 2^{-1}, 10^{-3}) 0.4723 (4)	0.4737 ± 0.0365 (2^{-7}, 0.1) 8.9304 (8)	**0.4496 ± 0.0369** (2^7, 2^0, 1.345) 0.3856 (1)	0.4625 ± 0.0337 (2^7, 2^{-1}, 1) 0.3609 (2)	0.4639 ± 0.0373 (2^8, 2^8, 2^0, 10^{-3}, 0.1) 0.5409 (3)	0.4680 ± 0.0510 (2^9, 2^9, 2^0, 10^{-3}, 0.1) 0.5612 (6)
Microsoft (563 × 5, 187 × 5)	0.5257 ± 0.0480 (2^9, 2^{-4}, 10^{-3}) 4.0485 (6)	0.5260 ± 0.0444 (2^9, 2^{-5}) 0.2215 (7)	0.5248 ± 0.0375 (2^0, 2^{-2}, 10^{-1}) 0.4818 (5)	0.5266 ± 0.0492 (2^{-7}, 0.1) 12.5356 (8)	0.5245 ± 0.0284 (2^9, 2^{-1}, 0.1) 0.3603 (4)	0.5214 ± 0.0512 (2^9, 2^{-1}, 0.1) 0.3256 (3)	**0.5210 ± 0.0405** (2^9, 2^9, 2^{-1}, 10^{-3}, 0.1) 0.5407 (1.5)	**0.5210 ± 0.0428** (2^9, 2^9, 2^{-2}, 10^{-3}, 1) 0.5646 (1.5)
Redhat (563 × 5, 187 × 5)	0.6485 ± 0.0837 (2^9, 2^{-5}, 10^{-1}) 4.0642 (6)	0.6751 ± 0.0776 (2^9, 2^{-5}) 0.2147 (8)	0.6550 ± 0.0948 (2^2, 2^{-2}, 10^{-1}) 0.4948 (7)	0.6484 ± 0.1101 (2^{-9}, 1.345) 18.0808 (5)	0.6462 ± 0.1049 (2^9, 2^{-1}, 1) 0.3762 (4)	0.6435 ± 0.1149 (2^9, 2^{-1}, 1) 0.3426 (3)	**0.6373 ± 0.0567** (2^9, 2^9, 2^{-1}, 10^{-3}, 1) 0.5565 (1)	0.6421 ± 0.0767 (2^9, 2^9, 2^{-1}, 10^{-3}, 1) 0.5607 (2)
Average rank	5.2143	5.8095	4.7619	6.5	3.0952	3.6190	3.1667	3.8333

proposed methods are robust and efficient learning methods for noisy datasets.

4.2 Benchmark datasets

In this subsection, we present the results of comparative analysis of SVR, LS-SVR, TSVR, RHSVR, FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR using both the linear and Gaussian kernels by performing experiments on 21 benchmark datasets. They are hydraulic actuator dataset [14, 33]; Pyrim, Pollution, Concrete CS, Sunspots, Servo, Triazines, Machine CPU, Wisconsin B.C., AutoMPG, Boston, Forestfires from UCI repository; Bodyfat, NO2 from http://lib.stat.cmu.edu/datasets/NO2.dat; and the financial time series datasets Citigroup, Google, IBM, Standard & Poor 500 (SNP500), Intel, Microsoft, Redhat from http://finance.yahoo.com.

For our first experimental study, we considered the hydraulic actuator dataset [14, 33] which is often used as a benchmark dataset for nonlinear system identification. It consists of 1024 pairs of valves $(u(t), y(t))$ where $u(t)$ is the size of the valve through which oil flows into the actuator and $y(t)$ is the oil pressure. By taking the input as $\mathbf{x}(t) = (y(t-1), y(t-2), \quad y(t-3), u(t-1), u(t-2))^t$ with its output being $y(t)$, we obtain 1021 samples of the form: $(\mathbf{x}(t), y(t))$ [14, 33]. Regarding the financial time series datasets considered, i.e., the stock index of Citigroup, Google, IBM, SNP500, Intel, Microsoft and RedHat, 755 closing prices starting from January 01, 2006, to December 31, 2008, are selected. The current stock index value is predicted using five previous values, and thus 750 samples are obtained.

Experiments are performed after normalizing the original data by taking: $\bar{x}_{ij} = \frac{x_{ij} - x_j^{\min}}{x_j^{\max} - x_j^{\min}}$, so that $\bar{x}_{ij}$ becomes the normalized value corresponding to the jth attribute value x_{ij} of the input example $\mathbf{x}_i$ wherein $x_j^{\min} = \min_{i=1,\ldots,m} x_{ij}$ and $x_j^{\max} = \max_{i=1,\ldots,m} x_{ij}$ denote the minimum and maximum values of the jth attribute value over all input examples $\mathbf{x}_i$.

In this numerical study, each dataset is randomly split into training and test sets in which 75% of the inputs are taken for training and the rest for testing. We randomly selected 20% of the training samples and polluted them with additive Gaussian noise with mean 0 and standard deviation 0.5. As in the study on synthetic datasets, the optimal parameter values are obtained by tenfold cross-validation procedure. To avoid biased comparisons, ten independent trials are performed. The results of the average test accuracy, the standard deviation and the average learning time for the linear and Gaussian kernels are summarized in Tables 5 and 6, respectively. Further, methods are ranked according to the accuracy obtained for every dataset and their average ranks are reported. It can be seen from Tables 5 and 6 that the test accuracy for the nonlinear case is, in general, better than the linear case and, however, faster learning speed is achieved for the linear case.

From the average ranks for the linear case reported in Table 5, it can be seen that the proposed algorithms FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR perform much better than the remaining algorithms with FHTSVR and RHSVR having the best and worst accuracies. However, it is worth to analyze their performances statistically. With eight algorithms and 21 datasets, we employ Friedman statistics. Under the null hypothesis that all the algorithms are equivalent, we compute [12] $\chi_F^2 \cong 65.4767$ and $F_F \cong 17.5721$, where F_F is distributed according to F-distribution with $(7, 7 \times 20) = (7, 140)$ degrees of freedom. Since 17.5721 is greater than 2.0756 which is the critical value of $F(7, 140)$ for the level of significance $\alpha = 0.05$, we reject the null hypothesis. Subsequently, as a post hoc test, we apply Nemenyi test for pairwise comparison of algorithms. According to Demsar [12], the critical value q_α for $\alpha = 0.10$ is 2.780 and the value of CD is $2.780\sqrt{\frac{8\times 9}{6\times 21}} \cong 2.1015$.

In terms of average ranks, the difference between: (1) the best and worst of the algorithms TSVR, FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR is $4.5476 - 2.9762 = 1.5714 < 2.1015$, and hence we conclude that the post hoc is unable to detect any significant difference between the algorithms; (2) the best of SVR, LS-SVR, RHSVR and the worst of FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR is $5.5476 - 3.4048 = 2.1428 > 2.1015$, which implies that the performance of FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR is better than SVR, LS-SVR and RHSVR; (3) the worst and the best of TSVR, SVR and LS-SVR is $5.9762 - 4.5476 = 1.4286 < 2.1015$, from which we can see that there is no significant difference between these algorithms; (4) the difference between TSVR and RHSVR is $7.3333 - 4.5476 = 2.7857 > 2.1015$ so we conclude that TSVR is better than RHSVR; and (5) the worst and the best of SVR, LS-SVR and RHSVR is $7.3333 - 5.5476 = 1.7857 < 2.1015$, so there is no significant difference among the algorithms.

We next examine the results for Gaussian kernel shown in Table 6 where experiments were performed on the same real-world datasets considered for the linear case. Clearly, FHTSVR outperforms all the other kernel methods by showing the best accuracy six times and is followed by ε-FHTSVR five times, whereas TSVR shows the worst performance with only one time the best test accuracy.

However, the average rank of TSVR is found to be better than SVR, LS-SVR and RHSVR. To avoid any biasness, it is proposed to analyze the results of Table 6 statistically by performing Friedman test and Nemenyi test. Under the null hypothesis that all the algorithms are equivalent, we compute $\chi_F^2 = 39.4261$ and $F_F = 7.33$, where F_F is distributed according to F-distribution with $(7, 7 \times 20) = (7, 140)$ degrees of freedom. From statistical table, $F(7, 140) = 2.0756$ for the level of significance $\alpha = 0.05$. Since $F_F > F(7, 140)$, we reject the null hypothesis. Subsequently, we proceed with Nemenyi post hoc test. From Demsar [12], at $\alpha = 0.10$, the value of CD is ≈ 2.1015. Again from Table 6, the difference between the average ranks of: (1) the best and the worst of TSVR, FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR is: $4.7619 - 3.0952 = 1.6667 < 2.1015$, from which we conclude that the post hoc test could not detect significant difference between the algorithms; (2) FHTSVR and the best of the algorithms SVR, LS-SVR and RHSVR is: $5.2143 - 3.0952 = 2.1191 > 2.1015$, which implies that the performance of FHTSVR is better than SVR, LS-SVR and RHSVR; (3) the best of TSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR, and the algorithm SVR is: $5.2143 - 3.1667 = 2.0476 < 2.1015$, from which we conclude that the post hoc test could not detect significant difference between the algorithms; (4) the worst of NHTSVR and ε-FHTSVR and the best of LS-SVR and RHSVR is: $5.8095 - 3.6190 = 2.1905 > 2.1015$, which implies that the performance of NHTSVR and ε-FHTSVR is better than LS-SVR and RHSVR; (5) the best of TSVR and ε-NHTSVR and LS-SVR is: $5.8095 - 3.8333 = 1.9762 < 2.1015$, from which we conclude that the post hoc test could not detect significant difference between the algorithms; (6) ε-NHTSVR and RHSVR is: $6.5 - 3.8333 = 2.6667 > 2.1015$, from which we conclude that ε-NHTSVR performs better than RHSVR; and (7) the best and the worst of SVR, LS-SVR, TSVR and RHSVR is: $6.5 - 4.7619 = 1.7381 < 2.1015$, from which we conclude that the test could not find significant difference between the three algorithms.

Because of the better average ranks shown by the proposed methods and by the above study of the statistical tests for comparisons of more algorithms on multiple datasets corrupted by noise and outliers, we conclude the superiority of FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR in terms of test accuracy and robustness. Again one can observe from Tables 5 and 6 that the training time of FHTSVR, NHTSVR, ε-FHTSVR and ε-NHTSVR remains low in comparison with SVR, TSVR and RHSVR. However, as expected, among all the algorithms, LS-SVR shows the least training time.

5 Conclusion and future work

In this paper, we have proposed novel robust regularized twin support vector regression methods based on Huber and ε-insensitive Huber loss functions in simple form with the aim of obtaining efficient and robust regression learning methods showing less sensitivity to large noise. Our formulation leads to solving a pair of strongly convex minimization problems in primal whose solutions are obtained by functional iterative and Newton–Armijo methods. The finite convergence of Newton–Armijo method has been proved. Our proposed methods have been tested on eight synthetic datasets polluted by four types of noise and by another ten synthetic datasets under the influence of asymmetric noise and outliers whose results have been compared with SVR, LS-SVR, TSVR and RHSVR. Additionally, experiments have been performed on 21 benchmark datasets corrupted by noise. Results demonstrate that our proposed methods are more efficient than traditional learning methods considered and in addition robust to noise and outliers present in the data space. Further, among the methods proposed, the best performance is shown by FHTSVR. The main difficulty of the proposed methods is the determination of the extra model parameters. However, we recall that as an important contribution, our proposed models give far more robust estimates than the quadratic loss and ε-insensitive loss-based models. We conclude that our proposed methods are efficient and attractive robust learning methods for regression showing insensitivity to noise and outliers present in the training samples. Future work includes the possible application of the proposed methods to real-world problems, like [15].

Acknowledgements The authors are extremely thankful to the reviewers for their comments. Mr. Subhash Chandra Prasad acknowledges the financial assistance awarded by Rajiv Gandhi National Fellowship, Government of India.

References

1. Balasundaram S, Gupta D (2014) Training Lagrangian twin support vector regression via unconstrained convex minimization. Knowl Based Syst 59:85–96
2. Balasundaram S, Gupta D, Prasad SC (2017) A new approach for training Lagrangian twin support vector machine via unconstrained convex minimization. Appl Intell 46:124–134
3. Balasundaram S, Tanveer M (2013) On Lagrangian twin support vector regression. Neural Comput Appl 22(Suppl. 1):S257–S267
4. Balasundaram S, Meena Y (2019) On robust regularized support vector regression in primal with asymmetric Huber loss. Neural Process Lett 49:1399–1431
5. Chapelle O (2007) Training a support vector machine in the primal. Neural Comput 19(5):1155–1178

6. Chen C, Li Y, Yan C, Liu G (2017) Least absolute deviation-based robust support vector regression. Knowl Based Syst 131:183–194
7. Chen C, Yan C, Zhao N, Guo B, Liu G (2017) A robust algorithm of support vector regression with a trimmed Huber loss function in the primal. Soft Comput 21(8):5235–5243
8. Chu W, Keerthi SS, Ong CJ (2004) Bayesian support vector regression using a unified loss function. IEEE Trans Neural Netw 15(1):29–44
9. Chuang CC, Lee ZJ (2011) Hybrid robust support vector machines for regression with outliers. Appl Soft Comput 11:64–72
10. Chuang CC, Su SF, Jeng JT, Hsiao CC (2002) Robust support vector regression networks for function approximation with outliers. IEEE Trans Neural Netw 13(6):1322–1330
11. Cristianini N, Shawe-Taylor J (2000) An introduction to support vector machines and other kernel based learning method. Cambridge University Press, Cambridge
12. Demsar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
13. Fung G, Mangasarian OL (2003) Finite Newton method for Lagrangian support vector machine. Neurocomputing 55:39–55
14. Gretton A, Doucet A, Herbrich R, Rayner PJW, Scholkopf B (2001) Support vector regression for black-box system identification. In: Proceedings of the 11th IEEE workshop on statistical signal processing
15. Guitton A, Symes WW (2003) Robust inversion of seismic data using the Huber norm. Geophysics 68(4):1310–1319
16. Guyon I, Weston J, Barnhill S, Vapnik V (2002) Gene selection for cancer classification using support vector machine. Mach Learn 46:389–422
17. Hao P-Y (2017) Pairing support vector algorithm for data regression. Neurocomputing 225:174–187
18. Hiriart-Urruty J-B, Strodiot JJ, Nguyen VH (1984) Generalized Hessian matrix and second-order optimality conditions for problems with C^{L1} data. Appl Math Optim 11:43–56
19. Huang X, Shi L, Pelckmans K, Suykens JAK (2014) Asymmetric v-tube support vector regression. Comput Stat Data Anal 77:371–382
20. Huber PJ, Ronchetti EM (2009) Robust statistics, 2nd edn. Wiley, New York
21. Lee Y-J, Hsieh W-F, Huang C-M (2005) ε-SSVR: a smooth support vector machine for ε-insensitive regression. IEEE Trans Knowl Data Eng 17(5):678–684
22. Jayadeva, Khemchandani R, Chandra S (2007) Twin support vector machines for pattern classification. IEEE Trans Pattern Anal Mach Intell 29(5):905–910
23. Mangasarian OL (1995) Parallel gradient distribution in unconstrained optimization. SIAM J Control Optim 33(6):1916–1925
24. Mangasarian OL, Musicant D (2000) Robust linear and support vector regression. IEEE Trans Pattern Anal Mach Intell 22(9):950–955
25. Mangasarian OL, Wild EW (2006) Multisurface proximal support vector classification via generalized eigenvalues. IEEE Trans Pattern Anal Mach Intell 28(1):69–74
26. Min JE, Lee YC (2005) Bankruptcy prediction using optimal choice of kernel function parameters. Expert Syst Appl 28(4):603–614
27. Osuna F, Freund R, Girosi F (1997) Training support vector machines: an application to face detection. In: Proceedings of computer vision and pattern recognition, pp 130–136
28. Peng X (2010) TSVR: an efficient twin support vector machine for regression. Neural Netw 23(3):365–372
29. Peng X (2011) TPMSVM: a novel twin parametric-margin support vector machine for pattern recognition. Pattern Recogn 44:2678–2692
30. Peng X, Xu D, Shen J (2014) A twin projection support vector machine for data regression. Neurocomputing 138:131–141
31. Rastogi R, Anand P, Chandra S (2017) A v-twin support vector machine based regression with automatic accuracy control. Appl Intell 46(3):670–683
32. Shao Y-H, Zhang C-H, Yang Z-M, Jing L, Deng N-Y (2013) An ε-twin support vector machine for regression. Neural Comput Appl 23(1):175–185
33. Sjoberg J, Zhang Q, Ljung L, Berveniste A, Delyon B, Glorennec P, Hjalmarsson H, Juditsky A (1995) Nonlinear black-box modeling in system identification: a unified overview. Automatica 31:1691–1724
34. Suykens JAK, De Brabanter J, Lukas L, Vandewalle J (2002) Weighted least squares support vector machines: robustness and sparse approximation. Neurocomputing 48(1):85–105
35. Suykens JAK, Gestel Van, De Brabanter J, De Moor B, Vandewalle J (2002) Least squares support vector machines. World Scientific, Singapore
36. Vapnik VN (2000) The nature of statistical learning theory, 2nd edn. Springer, New York
37. Wang X, Tan L, He L (2014) A robust least squares support vector machine for regression and classification with noise. Neurocomputing 140:41–52
38. Wang Z, Shao Y-H, Bai L, Deng N-Y (2015) Twin support vector machine for clustering. IEEE Trans Neural Netw Syst 26(10):2583–2588
39. Yang Z-M, Hua X-Y, Shao Y-H, Ye Y-F (2016) A novel parametric-insensitive nonparallel support vector machine for regression. Neurocomputing 171:649–663
40. Ye YF, Bai L, Hua XY, Shao YH, Wang Z, Deng NY (2016) Weighted Lagrange ε-twin support vector regression. Neurocomputing 197:53–68
41. Zhao Y, Sun J (2008) Robust support vector regression in the primal. Neural Netw 21:1548–1555
42. Zhao Y, Sun J (2010) Robust truncated support vector regression. Expert Syst Appl 37(7):5126–5133
43. Zhong P (2012) Training robust support vector regression with smooth non-convex loss function. Optim Methods Softw 27(6):1039–1058
44. Zhu J, Hoi SCH, Lyu MRT (2008) Robust regularized kernel regression. IEEE Trans Syst Man Cybern B Cybern 38(6):1639–1644