Neural Comput & Applic (2012) 21:399–407
DOI 10.1007/s00521-011-0525-6

ORIGINAL ARTICLE

Training twin support vector regression via linear programming

Ping Zhong · Yitian Xu · Yaohong Zhao

Received: 17 February 2010 / Accepted: 12 January 2011 / Published online: 4 February 2011

Abstract This paper improves the recently proposed twin support vector regression (TSVR) by formulating it as a pair of linear programming problems instead of quadratic programming problems. The use of 1-norm distance in the linear programming TSVR as opposed to the square of the 2-norm in the quadratic programming TSVR leads to the better generalization performance and less computational time. The effectiveness of the enhanced method is demonstrated by experimental results on artificial and benchmark datasets.

Keywords Support vector machine · Twin support vector machine · Regression · Nonparallel planes

1 Introduction

Support vector machine (SVM) [5, 27, 28] has been an excellent tool for classification and regression over the past decade as a modern machine learning approach, which enjoys strong theoretical foundations and successes in many real-world applications.

Although SVM owns better generalization classification ability compared with other machine learning methods like artificial neural network (ANN), its training cost is expensive, i.e., $O(n^3)$, where n is the total size of training data. So far, many fast algorithms have been presented. Traditionally, SVM is trained by using decomposition techniques such as SMO [13, 23] , chunking [21], SVM^{light} [11], and LIBSVM [3], which solve the dual problems by optimizing a small subset of the variables each iteration. Another kind of fast algorithm is the least squares SVM which was proposed by using equality constraints instead of inequality constraints [25]. All the above classifiers discriminate a sample by determining in which half space it lies. Mangasarian and Wild [19] first proposed a classification method named generalized eigenvalue proximal support vector machine (GEPSVM) by the proximity of the data to one of two nonparallel planes. Then Jayadeva, Khemchandani, and Chandra [10], proposed a twin support vector machine (TWSVM) classifier for classification in the spirit of GEPSVM. The formulation of TWSVM is very much similar to a classical SVM except that it aims at generating two nonparallel planes such that each plane is closer to one class and as far as possible from the other. TWSVM has become one of the popular methods because of its low computational complexity, see e.g. [8, 9, 15, 16].

As for support vector regression (SVR), there exist some corresponding fast algorithms such as SMO [24], LSSVR [26], smooth SVR [18], geometric method [1], heuristic training [29], and so on. All those methods for SVR have at least two groups of constraints, each of which ensures that more training samples locate within the given ϵ-insensitive tube as far as possible. Recently, Peng [22] proposed a twin support vector regression (TSVR) in the spirit of TWSVM. The TSVR aims at generating two functions such that each one determines the ϵ-insensitive down- or up-bound of the unknown regressor. For this purpose, it solves two smaller-sized quadratic programming problems (QPPs) instead of solving a large one as in the usual SVR. Although TSVR achieves good performance, it is less robust because the square of the 2-norm distance of the residuals is sensitive to the large errors. This fact motivates us to formulate TSVR as two linear programming problems by using the 1-norm distance. In order to distinguish our algorithm with that in [22], we term our algorithm as LPTSVR (linear

P. Zhong (✉) · Y. Xu · Y. Zhao
College of Science, China Agricultural University,
100083 Beijing, China
e-mail: pingsunshine@yahoo.com.cn

programming TSVR) and the algorithm in [22] as QPTSVR (quadratic programming TSVR). The use of 1-norm distance in the LPTSVR as opposed to the square of the 2-norm in the QPTSVR leads to the robustness, which makes the LPTSVR own the better generalization ability. Further, the LPTSVR allows us to solve twin linear programs as opposed to twin quadratic programs in the QPTSVR. We conclude results in Sect. 5, which demonstrate that linear programs are much more computationally efficient than quadratic programs.

The outline of the paper is as follows. A brief introduction of SVRs is given in Sect. 2. Section 3 presents the LPTSVR, and we extend it to the nonlinear cases in Sect. 4. In Sect. 5, the performance of the LPTSVR is compared with other SVRs. Section 6 concludes the paper.

2 Brief introduction of SVRs

2.1 The SVR

Let the learning samples to be denoted by a set of row vectors A_i $(i = 1, \ldots, n)$, where $A_i = (x_{i1}, x_{i2}, \ldots, x_{id})$. Let $A = [A_1; A_2; \ldots; A_n]$ represent the vertical concatenation of the vectors A_i. Let $Y = [y_1; y_2; \ldots; y_n]$ denote the response vector of training data points.

The SVR seeks to estimate a linear function $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ that tolerates a small error in fitting the given learning data, where $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. This can be obtained by using the ϵ-insensitive loss function, and any error smaller than ϵ is ignored. Applying the idea of SVM, the function $f(\boldsymbol{x})$ is made as flat as possible. The SVR can be formulated as the following QPP:

$$\begin{array}{ll} \min & \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + C(\boldsymbol{e}^\top \boldsymbol{\xi}_1 + \boldsymbol{e}^\top \boldsymbol{\xi}_2) \\ \text{s.t.} & Y - (A\boldsymbol{w} + \boldsymbol{e}b) \le \epsilon \boldsymbol{e} + \boldsymbol{\xi}_1, \quad \boldsymbol{\xi}_1 \ge \boldsymbol{0}, \\ & (A\boldsymbol{w} + \boldsymbol{e}b) - Y \le \boldsymbol{e}\epsilon + \boldsymbol{\xi}_2, \quad \boldsymbol{\xi}_2 \ge \boldsymbol{0}, \end{array} \tag{1}$$

where $C > 0$ is the regularization parameter that balances the tradeoff between the fitting errors and the flatness of the function, $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ are the slack vectors reflecting whether the samples locate into the ϵ-tube or not, $\boldsymbol{e}$ is the vector of ones of appropriate dimensions. The solution of (1) can be obtained by finding the saddle point of the Lagrange function. Then, the decision function is of the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} (\alpha_i^* - \alpha_i) \boldsymbol{x}_i^\top \boldsymbol{x} + b, \tag{2}$$

where $\boldsymbol{\alpha}, \boldsymbol{\alpha}^*$ are Lagrange multipliers that satisfy $\boldsymbol{\alpha}_i \boldsymbol{\alpha}_i^* = 0,\ i = 1, \ldots, n$.

The complexity of solving the dual QPP is $O(n^3)$. As the number of training samples increases, the time needed to train the SVR estimator also increases.

2.2 The LSSVR

Recall that the SVR requires large training time as the training samples increase, Suyken et al. [26] proposed the least squares version of SVR termed as LSSVR to improve it. The formulation of LSSVR is as follows:

$$\begin{array}{ll} \min & \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + \frac{C}{2}\boldsymbol{\xi}^\top \boldsymbol{\xi} \\ \text{s.t.} & Y = A\boldsymbol{w} + \boldsymbol{e}b + \boldsymbol{\xi}. \end{array} \tag{3}$$

LSSVR finds the regression function by solving a set of n linear equations. Its computational complexity depends on the algorithm used for its solution. For example, the Gaussian elimination technique leads to a complexity of $O(n^3)$, whereas for the conjugate gradient method, it is less than $O(n^3)$. In comparison with the SVR, the LSSVR is not sparseness. In addition, it is less robust because of the sensitivity of sum squared error. In recent years, considerable attentions have been paid to these limitations, see e.g., [12, 14, 17, 30, 31].

2.3 The QPTSVR

The QPTSVR aims at finding two functions

$$f_1(\boldsymbol{x}) = \boldsymbol{w}_1^\top \boldsymbol{x} + b_1, \quad f_2(\boldsymbol{x}) = \boldsymbol{w}_2^\top \boldsymbol{x} + b_2, \tag{4}$$

such that each function determines one of the ϵ-insensitive down- or up-bound regressor. It is obtained by solving the following pair of QPPs

$$\begin{array}{ll} \min & \frac{1}{2}(Y - \boldsymbol{e}\epsilon_1 - (A\boldsymbol{w}_1 + \boldsymbol{e}b_1))^\top (Y - \boldsymbol{e}\epsilon_1 - (A\boldsymbol{w}_1 + \boldsymbol{e}b_1)) + C_1 \boldsymbol{e}^\top \boldsymbol{\xi}_1 \\ \text{s.t.} & Y - (A\boldsymbol{w}_1 + \boldsymbol{e}b_1) \ge \boldsymbol{e}\epsilon_1 - \boldsymbol{\xi}_1, \quad \boldsymbol{\xi}_1 \ge \boldsymbol{0}, \end{array} \tag{5}$$

$$\begin{array}{ll} \min & \frac{1}{2}(Y + \boldsymbol{e}\epsilon_2 - (A\boldsymbol{w}_2 + \boldsymbol{e}b_2))^\top (Y + \boldsymbol{e}\epsilon_2 - (A\boldsymbol{w}_2 + \boldsymbol{e}b_2)) + C_2 \boldsymbol{e}^\top \boldsymbol{\xi}_2 \\ \text{s.t.} & (A\boldsymbol{w}_2 + \boldsymbol{e}b_2) - Y \ge \boldsymbol{e}\epsilon_2 - \boldsymbol{\xi}_2, \quad \boldsymbol{\xi}_2 \ge \boldsymbol{0}, \end{array} \tag{6}$$

where C_1 and $C_2 > 0$ are regularization parameters. The end regressor is decided by the mean of these two functions:

$$f(\boldsymbol{x}) = \frac{1}{2}(f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})) = \frac{1}{2}(\boldsymbol{w}_1 + \boldsymbol{w}_2)^\top \boldsymbol{x} + \frac{1}{2}(b_1 + b_2). \tag{7}$$

The QPTSVR is comprised of a pair of QPPs such that each QPP determines the one of up- or down-bound function by using only one group of constraints compared with the standard SVR. Hence, the QPTSVR gives rise to two smaller-sized QPPs. As TWSVM, the QPTSVR is approximately four times faster than the SVR in theory.

3 The LPTSVR

Notice that the first term in the objective function of (5) or (6) is the sum of squared distances from the shifted function

$y = \boldsymbol{w}_1^\top \boldsymbol{x} + b_1 + \epsilon_1$ or $y = \boldsymbol{w}_2^\top \boldsymbol{x} + b_2 - \epsilon_2$ to the training points. That is, the residuals between the real values and the predicted values are penalized by the square of the 2-norm, which is sensitive to the large errors. By replacing the square of the 2-norm $\|Y - \boldsymbol{e}\epsilon_1 - (A\boldsymbol{w}_1 + \boldsymbol{e}b_1)\|_2^2$ or $\|Y + \boldsymbol{e}\epsilon_2 - (A\boldsymbol{w}_2 + \boldsymbol{e}b_2)\|_2^2$ in the quadratic program (5) or (6) with the 1-norm $\|Y - \boldsymbol{e}\epsilon_1 - (A\boldsymbol{w}_1 + \boldsymbol{e}b_1)\|_1$ or $\|Y + \boldsymbol{e}\epsilon_2 - (A\boldsymbol{w}_2 + \boldsymbol{e}b_2)\|_1$, we obtain the LPTSVR as follows.

$$\begin{array}{ll} \min & v_1 \boldsymbol{e}^\top \boldsymbol{\xi}_1 + \|Y - \boldsymbol{e}\epsilon_1 - (A\boldsymbol{w}_1 + \boldsymbol{e}b_1)\|_1 \\ \text{s.t.} & Y - (A\boldsymbol{w}_1 + \boldsymbol{e}b_1) \geq \boldsymbol{e}\epsilon_1 - \boldsymbol{\xi}_1, \quad \boldsymbol{\xi}_1 \geq \boldsymbol{0}, \end{array} \tag{8}$$

$$\begin{array}{ll} \min & v_2 \boldsymbol{e}^\top \boldsymbol{\xi}_2 + \|Y + \boldsymbol{e}\epsilon_2 - (A\boldsymbol{w}_2 + \boldsymbol{e}b_2)\|_1 \\ \text{s.t.} & (A\boldsymbol{w}_2 + \boldsymbol{e}b_2) - Y \geq \boldsymbol{e}\epsilon_2 - \boldsymbol{\xi}_2, \quad \boldsymbol{\xi}_2 \geq \boldsymbol{0}, \end{array} \tag{9}$$

where v_1 and $v_2 > 0$ are regularization parameters. The slack vector $\boldsymbol{\xi}_1$ or $\boldsymbol{\xi}_2$ is introduced to measure the error wherever the distance is closer than ϵ_1 or ϵ_2.

We note that the term $\|Y - \boldsymbol{e}\epsilon_1 - (A\boldsymbol{w}_1 + \boldsymbol{e}b_1)\|_1$ in (8) or $\|Y + \boldsymbol{e}\epsilon_2 - (A\boldsymbol{w}_2 + \boldsymbol{e}b_2)\|_1$ in (9) is easily converted to a linear term $\boldsymbol{e}^\top \boldsymbol{s}$ with the added constraint $-\boldsymbol{s} \leq Y - \boldsymbol{e}\epsilon_1 - (A\boldsymbol{w}_1 + \boldsymbol{e}b_1) \leq \boldsymbol{s}$ or $-\boldsymbol{s} \leq Y + \boldsymbol{e}\epsilon_2 - (A\boldsymbol{w}_2 + \boldsymbol{e}b_2) \leq \boldsymbol{s}$. Hence, the LPTSVR allows us to solve twin linear programs as opposed to quadratic programs. Further, the 1-norm distance is less sensitive to large errors. The use of 1-norm distance in the LPTSVR as opposed to the square of the 2-norm in the QPTSVR leads to the robustness to large noises.

As a toy example, for data generated by $y_i = x_i + \eta_i$ with x_i being simulated uniformly in [0,1] and η being Gaussian noise N $(0,0.01^2)$, we calculated the final regressors by the QPTSVR and the LPTSVR, respectively. For simplicity, we set $\epsilon_1 = \epsilon_2 = 0$. Moreover, we added large noises in the dataset to test the robustness of the two methods. Results are shown in Fig. 1. We can see from the plots that for the dataset without large noises, these two regressors almost overlap. However, for the dataset with large noises, the regressor calculated by the QPTSVR is drawn toward noises, while the regressor calculated by the LPTSVR is robust to noises.

4 Kernelizing LPTSVR

In order to extend the linear LPTSVR to the nonlinear case, we consider the following kernel-generated functions instead of linear functions.

$$f_1(\boldsymbol{x}) = K(\boldsymbol{x}^\top, A^\top)\boldsymbol{u}_1 + b_1, \; f_2(\boldsymbol{x}) = K(\boldsymbol{x}^\top, A^\top)\boldsymbol{u}_2 + b_2, \tag{10}$$

where $K(\cdot,\cdot)$ is a kernel representing the inner product in the feature space $\mathbb{H}$, i.e., $K(\boldsymbol{u}^\top, \boldsymbol{v}) = \varphi(\boldsymbol{u})^\top \varphi(\boldsymbol{v})$ with $\varphi(\cdot)$ being a nonlinear map from the input space R^d into the feature space $\mathbb{H}$. We use $K(A, B^\top)$ to denote the kernel matrix with $K_{ij} = K(A_i, B_j^\top)$.

We note that the linear functions (4) are recovered from (10) if we use the linear kernel $K(\boldsymbol{x}^\top, A^\top) = \boldsymbol{x}^\top A^\top$ and define $\boldsymbol{w}_1 = A^\top \boldsymbol{u}_1$ or $\boldsymbol{w}_2 = A^\top \boldsymbol{u}_2$. With this nonlinear kernel formulations, the mathematical programs for generating the nonlinear regressor become the following, upon kernelizing (8) and (9):

$$\begin{array}{ll} \min & v_1 \boldsymbol{e}^\top \boldsymbol{\xi}_1 + \boldsymbol{e}^\top \boldsymbol{s}_1 \\ \text{s.t.} & -\boldsymbol{s}_1 \leq Y - \boldsymbol{e}\epsilon_1 - (K(A, A^\top)\boldsymbol{u}_1 + \boldsymbol{e}b_1) \leq \boldsymbol{s}_1, \\ & Y - (K(A, A^\top)\boldsymbol{u}_1 + \boldsymbol{e}b_1) \geq \boldsymbol{e}\epsilon_1 - \boldsymbol{\xi}_1, \\ & \boldsymbol{\xi}_1 \geq \boldsymbol{0}, \end{array} \tag{11}$$

$$\begin{array}{ll} \min & v_2 \boldsymbol{e}^\top \boldsymbol{\xi}_2 + \boldsymbol{e}^\top \boldsymbol{s}_2 \\ \text{s.t.} & -\boldsymbol{s}_2 \leq Y + \boldsymbol{e}\epsilon_2 - (K(A, A^\top)\boldsymbol{u}_2 + \boldsymbol{e}b_2) \leq \boldsymbol{s}_2, \\ & (K(A, A^\top)\boldsymbol{u}_2 + \boldsymbol{e}b_2) - Y \geq \boldsymbol{e}\epsilon_2 - \boldsymbol{\xi}_2, \\ & \boldsymbol{\xi}_2 \geq \boldsymbol{0}. \end{array} \tag{12}$$

5 Experiments and discussion

To demonstrate the validity of the proposed LPTSVR, we compared it with the LSSVR and the QPTSVR on a collection of datasets, including a group of artificial datasets and seventeen benchmark datasets. All the regression methods were implemented on a PC with AMD 3600+(2.00 GHz) processor, 512 MB memory in Matlab 7.0 environment. As the machine learning tools, the performances of these algorithms depend on the choices of

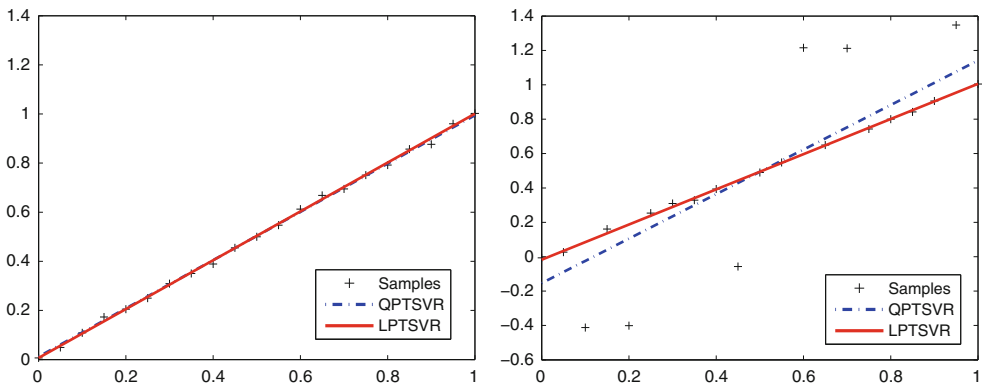

Fig. 1 Predictions of the QPTSVR with $C_1 = C_2 = 2^{-2}$ and the LPTSVR with $v_1 = v_2 = 2^{-2}$. The *left* graph shows the regressors calculated on the dataset without large noises. In the *right* graph, the regressors are calculated on the dataset with large noises

parameters. In our experiments, we set $v_1 = v_2$ and $\epsilon_1 = \epsilon_2$ in the LPTSVR, $C_1 = C_2$ and $\epsilon_1 = \epsilon_2$ in the QPTSVR to degrade the computational complexity of parameter selection. In order to evaluate the performance of the algorithms, the evaluation criteria are specified before presenting the experimental results. Let N be the number of testing samples, y_i and $\hat{y}_i$ be the real value and predicted value of sample $\boldsymbol{x}_i$, respectively. Denote $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ as the average value of $y_1, \ldots, y_N$. We use the following criteria for algorithm evaluation.

RMSE: Root mean squared error, which is defined as $\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$. It is commonly used as the deviation measurement between the real and predicted values. RMSE represents the fitting precision; the smaller RMSE is, the fitter the estimation is. However, when noises are also used as testing samples, too small value of RMSE probably speaks for overfitting of the regressor.

MAE: Mean absolute error, which is defined as $\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$. MAE is also a popular deviation measurement between the real and predicted values.

SSE/SST [22]: Ratio between the sum squared error $\mathrm{SSE} = \sum_{i=1}^{N}(\hat{y}_i - y_i)^2$ and the sum squared deviation of testing samples $\mathrm{SST} = \sum_{i=1}^{N}(y_i - \bar{y})^2$. SSE is in fact a variant of RMSE. SST reflects the underlying variance of the testing samples. In most cases, small SSE/SST means good agreement between estimations and real values.

SSR/SST [22]: Ratio between interpretable sum squared deviation $\mathrm{SSR} = \sum_{i=1}^{N}(\hat{y}_i - \bar{y})^2$ and SST. SSR reflects the explanation ability of the regressor. The larger SSR is, the more statistical information it catches from testing samples. To obtain smaller SSE/SST usually accompanies an increase in SSR/SST. However, the extremely small value of SSE/SST is in fact not good, for it probably means overfitting of the regressor. Therefore, a good estimator should strike balance between SSE/SST and SSR/SST.

To compare the CPU time and accuracies of three algorithms, we used "quadprog.m" function to realize the QPTSVR and the "inv.m" function to implement the LSSVR. For the LPTSVR, we first applied the "linprog.m" function in Matlab to realize it. We found that the LPTSVR needed less time on moderate and large-scale datasets. However, for the small-scale datasets, it was not the case. In order to improve the speed, we employed Newton method for 1-norm SVM in [20] to speed up it. In addition, we normalized the training data into the closed span [0,1] and mapped back to the original space when we calculate accuracies of algorithms.

5.1 Artificial datasets

The regressions of sinc function $y = \frac{\sin\pi x}{\pi x}, \quad x \in [-5, 5]$ were tested to reflect the performance of the LPTSVR. Training data points were perturbed by some different kinds of noises, which include the Gaussian noises and the uniformly distributed noises. Specially, the following kinds of training samples $(x_i, y_i), i = 1, \ldots, n$ were generated:

$$y_i = \frac{\sin\pi x_i}{\pi x_i} + \eta_i, \quad x \sim U[-5,5], \quad \eta_i \sim N(0, 0.15^2), \tag{13}$$

$$y_i = \frac{\sin\pi x_i}{\pi x_i} + \eta_i, \quad x \sim U[-5,5], \quad \eta_i \sim N(0, 0.3^2), \tag{14}$$

$$y_i = \frac{\sin\pi x_i}{\pi x_i} + \eta_i, \quad x \sim U[-5,5], \quad \eta_i \sim U[0, 0.15], \tag{15}$$

$$y_i = \frac{\sin\pi x_i}{\pi x_i} + \eta_i, \quad x \sim U[-5,5], \quad \eta_i \sim U[0, 0.3], \tag{16}$$

where N $(0,d^2)$ represents the Gaussian random variable with zero means and variance d^2, and $U[a, b]$ represents the uniformly random variable in $[a, b]$.

In order to avoid of biased comparisons, for each kind of noises, we randomly generated ten independent groups of noisy samples which respectively consists of 350 training samples and 500 test samples. The test data are uniformly sampled from the objective sinc function without any noise. Table 1 shows the average accuracies of the LPTSVR, QPTSVR, and LSSVR with ten independent runs, in which types I, II, III, and IV represent the four different types of noises (13–16). It has been shown that the LPTSVR obtains the smallest RMSE and MAE among the three algorithms for three types of noises (types I, II, and

Table 1 Comparisons of LSSVR, QPTSVR, and LPTSVR on Sinc datasets with different types of noises

Noise	Algorithm	RMSE	MAE	SSE/SST	SSR/SST	Time (s)
Type I	LSSVR	0.0291	0.0239	0.0099	0.9881	0.128
	QPTSVR	0.0322	0.0267	0.0121	1.0294	46.231
	LPTSVR	0.0283	0.0235	0.0095	1.0273	43.473
Type II	LSSVR	0.0554	0.0456	0.0359	0.9795	0.128
	QPTSVR	0.0643	0.0533	0.0483	1.0831	46.128
	LPTSVR	0.0539	0.0445	0.0339	1.0774	41.812
Type III	LSSVR	0.0759	0.0755	0.0662	1.0569	0.127
	QPTSVR	0.0746	0.0742	0.0640	1.0599	45.806
	LPTSVR	0.0755	0.0749	0.0655	1.0659	45.586
Type IV	LSSVR	0.1517	0.1510	0.2645	1.2459	0.133
	QPTSVR	0.1507	0.1498	0.2609	1.2542	46.128
	LPTSVR	0.1506	0.1495	0.2611	1.2549	42.336

Table 2 Results of LSSVR (I), QPTSVR (II), and LPTSVR (III) on benchmark datasets with linear kernel

Dataset Samp. × Feat.	Alg.	RMSE	MAE	SSE/SST	SSR/SST	Time(s)	$C(\nu)$	ϵ
BH (506 × 14)	I	4.9373 ± 2.3970	3.7216 ± 1.5869	0.5921 ± 0.3054	0.8208 ± 0.5627	1.6406	2^{-1}	–
	II	5.0500 ± 2.6524	3.7947 ± 1.8626	0.6527 ± 0.4123	1.0523 ± 0.7831	1.0850E+3	2^{-3}	10^{-3}
	III	4.8847 ± 2.7639	3.6114 ± 1.8629	0.6035 ± 0.4407	0.9017 ± 0.5276	464.2813	2^{-4}	10^{-2}
MCPU (209 × 7)	I	60.7936 ± 42.5666	38.6041 ± 19.0112	0.5702 ± 0.4361	1.3121 ± 1.0105	0.3125	2^{-1}	–
	II	64.2079 ± 40.8823	41.4762 ± 18.6620	0.7328 ± 0.5860	1.3670 ± 1.0531	73.2969	2^{-1}	10^{-3}
	III	58.6864 ± 41.0596	36.8308 ± 15.8253	0.6147 ± 0.4917	1.1880 ± 0.9870	52.4688	2^{-4}	0.2
Con. CS (1,030 × 9)	I	39.7910 ± 9.2708	31.9146 ± 7.1855	0.2609 ± 0.1733	1.0245 ± 0.2073	11.0156	2^{6}	–
	II	39.7922 ± 9.2581	31.9088 ± 7.1811	0.2615 ± 0.1755	1.0264 ± 0.2071	6.6979E+3	2^{7}	10^{-3}
	III	39.6472 ± 9.4421	32.1696 ± 8.8168	0.2621 ± 0.1724	1.0781 ± 0.2580	4.9774E+3	2^{4}	10^{-3}
Wis. BC (194 × 33)	I	31.9720 ± 6.5073	27.2541 ± 5.9645	3.6605 ± 6.7018	2.8704 ± 6.5833	0.2500	2^{-2}	–
	II	33.2890 ± 7.5830	27.9749 ± 7.4515	4.6834 ± 9.5912	3.9263 ± 9.4023	87.0781	2^{1}	10^{-3}
	III	31.9237 ± 7.6301	27.2410 ± 6.6008	3.4446 ± 6.1084	2.6301 ± 5.9900	69.5625	2^{1}	0.2
AutoMPG (392 × 8)	I	3.4234 ± 0.8778	2.6980 ± 0.6228	0.3453 ± 0.1791	0.9020 ± 0.4575	0.1875	2^{5}	–
	II	3.4214 ± 0.8856	2.6960 ± 0.6270	0.3455 ± 0.1810	0.9021 ± 0.4557	449.8281	2^{-4}	10^{-3}
	III	3.3674 ± 1.0677	2.6348 ± 0.7545	0.3403 ± 0.2186	0.8040 ± 0.3725	361.3125	2^{1}	0.2
AutoPrice (159 × 16)	I	2812.4 ± 1025.7	2081.6 ± 732.5	0.8296 ± 1.2464	1.6967 ± 1.8700	0.0938	2^{0}	–
	II	3124.9 ± 1378.7	2421.5 ± 1015.6	0.6572 ± 0.4067	1.5087 ± 1.1525	34.3125	2^{1}	10^{-3}
	III	2781.6 ± 1147.7	2105.8 ± 873.4	0.6026 ± 0.5365	1.5062 ± 1.1515	15.0469	2^{2}	0.2
Servo (167 × 5)	I	1.1277 ± 0.2132	0.9229 ± 0.1121	1.5769 ± 2.8727	1.8426 ± 3.8055	0.0938	2^{3}	–
	II	1.1261 ± 0.2178	0.9111 ± 0.1184	1.5596 ± 2.8472	1.7588 ± 3.7600	32.3594	2^{-1}	10^{-3}
	III	1.1272 ± 0.2151	0.9195 ± 0.1128	1.5560 ± 2.8193	1.8024 ± 3.7317	32.3125	2^{-6}	10^{-2}
Pyrim (74 × 28)	I	0.0889 ± 0.0605	0.0638 ± 0.0317	1.6247 ± 2.2491	1.9330 ± 2.5221	0.1406	2^{0}	–
	II	0.1348 ± 0.0757	0.0824 ± 0.0308	6.6566 ± 9.7054	7.6557 ± 10.8568	5.1094	2^{1}	10^{-3}
	III	0.0869 ± 0.0495	0.0632 ± 0.0256	1.9572 ± 3.0902	2.4882 ± 3.5123	4.2500	2^{5}	0.2
Diabetes (43 × 3)	I	0.5763 ± 0.1565	0.4647 ± 0.1211	0.9446 ± 0.3622	0.4632 ± 0.5448	0.1250	2^{1}	–
	II	0.5769 ± 0.1658	0.4564 ± 0.1201	1.0049 ± 0.5655	0.6713 ± 0.7615	2.0625	2^{2}	10^{-3}
	III	0.5748 ± 0.1459	0.4634 ± 0.1103	1.0107 ± 0.5609	0.6712 ± 0.8658	0.9844	2^{1}	0.2
Triazines (186 × 61)	I	0.1395 ± 0.0220	0.1033 ± 0.0166	0.9137 ± 0.2467	0.3323 ± 0.2420	0.3125	2^{-2}	–
	II	0.1512 ± 0.0173	0.1093 ± 0.0105	1.0943 ± 0.3221	0.6520 ± 0.4128	69.1094	2^{-1}	10^{-2}
	III	0.1432 ± 0.0213	0.1077 ± 0.0137	1.0055 ± 0.3880	0.5251 ± 0.3487	52.1094	2^{-1}	10^{-2}
Chwirut1 (214 × 2)	I	12.7428 ± 1.8973	11.1487 ± 1.5396	0.3093 ± 0.0542	0.7436 ± 0.1324	0.1875	2^{2}	–
	II	12.7211 ± 1.9365	11.0802 ± 1.5460	0.3079 ± 0.0531	0.7271 ± 0.1262	60.8750	2^{-1}	10^{-3}
	III	12.7426 ± 1.9748	11.0723 ± 1.5643	0.3087 ± 0.0535	0.7297 ± 0.1282	33.3125	2^{4}	0.2
TH (49 × 2)	I	0.3046 ± 0.2071	0.2365 ± 0.1458	46.3409 ± 81.4538	44.5929 ± 81.7178	0.0156	2^{7}	–
	II	0.2938 ± 0.2238	0.2152 ± 0.1568	23.5557 ± 35.3689	21.7957 ± 34.8546	2.3281	2^{-3}	10^{-3}
	III	0.2911 ± 0.2255	0.2112 ± 0.1670	19.3777 ± 32.0584	17.3328 ± 28.6244	1.5781	2^{-4}	0.2
Mg (1,385 × 7)	I	0.1472 ± 0.0099	0.1176 ± 0.0078	0.4252 ± 0.0577	0.5885 ± 0.0444	24.5156	2^{7}	–
	II	0.1472 ± 0.0099	0.1176 ± 0.0078	0.4252 ± 0.0578	0.5886 ± 0.0442	1.5908E+4	2^{-1}	10^{-3}
	III	0.1475 ± 0.0110	0.1173 ± 0.0090	0.4275 ± 0.0647	0.5809 ± 0.0458	4.699E+3	2^{1}	10^{-2}
Con. S (103 × 8)	I	7.9549 ± 2.0348	6.7652 ± 1.6548	1.3565 ± 0.5072	0.5403 ± 0.5709	0.0781	2^{-3}	–
	II	8.3266 ± 2.0938	7.0944 ± 1.9409	1.7978 ± 1.5395	1.0796 ± 1.0415	9.4375	2^{-7}	10^{-3}
	III	8.0611 ± 1.8092	6.8176 ± 1.5149	1.5026 ± 0.8145	0.8114 ± 0.8330	4.8438	2^{1}	0.2
Bodyfat (252 × 15)	I	0.0020 ± 0.0023	0.0010 ± 0.0005	0.0306 ± 0.0678	0.9745 ± 0.0956	0.4219	2^{7}	–
	II	0.0019 ± 0.0024	7.3708E-4 ± 0.0006	0.0276 ± 0.0672	0.9718 ± 0.0869	155.2344	2^{-4}	10^{-3}
	III	0.0017 ± 0.0024	6.3148E-4 ± 0.0006	0.0267 ± 0.0670	0.9701 ± 0.0819	126.6875	2^{-4}	10^{-2}

IV). This indicates that the LPTSVR is robust to noises. Besides, the LPTSVR derives the smallest SSE/SST values for the first two types of noises and the largest SSR/SST values for the last two types of noises. Table 1 also compares the training time for these three algorithms. It has been shown that the LSSVR is the fastest learning method among them, whereas the LPTSVR is faster than the QPTSVR.

5.2 Benchmark datasets

For further evaluation, we tested seventeen real-world benchmark datasets, including Boston Housing (BH), AutoMPG, Servo, Pyrimidines (Pyrim), Triazines, Concrete Slump (Con. S) from UCI repository [2], Concrete Compressive Strength (Con. CS), Pollution, Bodyfat from StatLib database[1], AutoPrice, Machine CPU (MCPU), Wisconsin Breast Cancer (Wis. BC), Diabetes from http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html, Mg[2], Chwirut1[3], Ozone[4], and Titanium Heat (TH) [6]. The Concrete Slump dataset has three output features. In our experiments, we merely used slump feature as the output.

First, we make the linear kernel comparisons of the LPTSVR, LSSVR, and QPTSVR. We evaluated the regression accuracy using 10-fold cross-validation. The optimal ϵ values for the LPTSVR and QPTSVR were searched in the range $\{0.001, 0.01, 0.1, 0.2\}$, and the optimal regularization parameters C (or ν) for the three algorithms were selected over the range $\{2^i | i = -7, \ldots, 7\}$. Table 2 lists the experimental results of the three algorithms on fifteen benchmark datasets. It consists of the mean values of RMSE, MAE, SSE/SST, SSR/SST on each dataset, the standard deviation values of the four measures on each dataset, the computation time, and the optimal parameter values.

Since the Friedman test with the corresponding post hoc tests is pointed out to be a simple, safe, and robust nonparametric test for comparison of more classifiers over multiple datasets [7], we use it to compare the performance of the three algorithms. Table 3 illustrates the average ranks of the three algorithms on RMSE, MAE, SSE/SST, and SSR/SST values. The computational process of the average rank of the three algorithms on RMSE values is shown in "Appendix" section as an example. First, we compare the RMSE performance of the three algorithms.

[1] Available from http://lib.stat.cmu.edu/datasets/.

[2] Available from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression.html.

[3] Available from http://www.itl.nist.gov/div898/strd/nls/nls-main.shtml.

[4] Available from http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/ozone.data

Table 3 Average ranks of LSSVR, QPTSVR, and LPTSVR with linear kernel

Algorithm	RMSE	MAE	SSE/SST	SSR/SST
LSSVR	2.167	2.233	1.8333	2.0667
QPTSVR	2.433	2.3	2.3667	1.5333
LPTSVR	1.4	1.467	1.8	2.4

We employ the Friedman test to check whether the measured average ranks are significantly different from the mean rank $R_j = 2$ expected under the null hypothesis:

$$\chi_F^2 = \frac{12 \times 15}{3 \times 4}\left[(2.167^2 + 2.433^2 + 1.4^2) - \frac{3 \times 4^2}{4}\right] = 8.6307$$

$$F_F = \frac{14 \times 8.6307}{15 \times 2 - 8.6307} = 5.6544$$

With three algorithms and 15 datasets, F_F is distributed according to the F distribution with (2,28) degrees of freedom. The critical value of $F(2,28)$ for $\alpha = 0.05$ is 3.340, so we reject the null hypothesis. We use the Nemenyi test for further pairwise comparison. According to [7], at $p = 0.10$, CD is $2.052\sqrt{\frac{3 \cdot 4}{6 \cdot 15}} = 0.7493$. Since the difference between the LPTSVR and the QPTSVR is larger than 0.7493 ($2.433 - 1.4 = 1.033 > 0.7493$), we can identify that the performance of the LPTSVR is better than that of the QPTSVR. In the same way, we see that the performance of the LPTSVR is better than that of the LSSVR ($2.167 - 1.4 = 0.767 > 0.7493$). Similarly, we obtain that the F_F value on MAE is 3.8160, which is larger than the critical value 3.340, so we reject the null hypothesis. From Table 3, we see that the LPTSVR performs significantly better than the QPTSVR ($2.3 - 1.467 = 0.833 > 0.7493$) and the LSSVR ($2.233 - 1.467 = 0.766 > 0.7493$). The F_F value on SSE/SST is 1.5751, which is smaller than the critical value 3.340 (or 2.503, which is the critical value of $F(2,28)$ for $\alpha = 0.10$), so there is no significant difference between the three algorithms. For SSR/SST, the F_F value is 3.3081 which is larger than 2.503, so we reject the null hypothesis. By the further post hoc test, we conclude that the performance of the QPTSVR is better than that of the LPTSVR.

As for the computation time, the LSSVR spends on the least CPU time among the three algorithms, while the LPTSVR needs the less CPU time than the QPTSVR.

For the nonlinear case, the RBF kernel $k(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\|\boldsymbol{x} - \boldsymbol{z}\|^2/\gamma)$ was chosen for all algorithm training. As the machine learning tools, the performance of algorithms seriously depends on the choice of parameters. As we know, the conventional exhaustive grid search method is prohibitively performed. So we take a simple variant instead, which may not find the same hyper-parameters as

Table 4 Results of LSSVR (I), QPTSVR (II), and LPTSVR (III) on benchmark datasets with RBF kernel

Dataset Samp. × Feat.	Alg.	RMSE	MAE	SSE/SST	SSR/SST	Time(s)	γ	C (ν)	ϵ
BH (506 × 14)	I	4.0293 ± 1.9732	2.9333 ± 1.0754	0.4272 ± 0.2773	0.8765 ± 0.3801	2.094	2^{1}	2^{2}	–
	II	3.9141 ± 1.9743	2.8831 ± 1.1484	0.4362 ± 0.3140	1.0496 ± 0.4055	5027.125	2^{6}	2^{-3}	10^{-3}
	III	3.8084 ± 2.0733	2.7891 ± 1.1591	0.4261 ± 0.3160	0.9718 ± 0.3924	1501.719	2^{7}	2^{1}	10^{-3}
MCPU (209 × 7)	I	67.8834 ± 59.0666	36.1499 ± 22.1940	0.5133 ± 0.3869	0.8388 ± 0.9181	0.5790	2^{0}	2^{3}	–
	II	53.7474 ± 36.9823	32.5153 ± 16.1998	0.4305 ± 0.3523	1.0499 ± 0.7989	81.0310	2^{6}	2^{-1}	10^{-3}
	III	52.5494 ± 38.3332	32.1576 ± 17.4324	0.4344 ± 0.3323	0.9659 ± 0.6671	76.4915	2^{3}	2^{-3}	10^{-3}
Con. CS (1,030 × 9)	I	36. 1796 ± 10.8120	28.4257 ± 9.1352	0.1953 ± 0.0842	0.9532 ± 0.1390	12.625	2^{0}	2^{3}	–
	II	37.8044 ± 12.3588	30.6511 ± 11.2172	0.2109 ± 0.0875	1.0174 ± 0.2018	10068.219	2^{7}	2^{-3}	10^{-3}
	III	37.1821 ± 12.8103	29.7158 ± 10.7025	0.2048 ± 0.0810	1.0494 ± 0.1616	6234.563	2^{4}	2^{-3}	10^{-3}
AutoMPG (392 × 8)	I	2.5197 ± 0.8509	1.8912 ± 0.5624	0.1974 ± 0.1434	0.8784 ± 0.1570	1.453	2^{0}	2^{4}	–
	II	2.7247 ± 0.8544	2.0471 ± 0.5815	0.2351 ± 0.1572	0.9523 ± 0.2125	1898.614	2^{3}	2^{-3}	10^{-3}
	III	2.6623 ± 0.7797	2.0242 ± 0.5567	0.2205 ± 0.1392	0.8297 ± 0.1734	907.938	2^{-1}	2^{-1}	10^{-3}
AutoPrice (159 × 16)	I	2855.8 ± 1316.8	2108.1 ± 927.4	0.4849 ± 0.3116	1.1709 ± 0.8639	0.140	2^{1}	2^{0}	–
	II	2812.0 ± 1387.3	2030.3 ± 927.6	0.4994 ± 0.3466	1.3298 ± 1.0109	45.282	2^{7}	2^{-3}	10^{-3}
	III	2710.0 ± 1274.1	1970.0 ± 900.2	0.4565 ± 0.3343	1.3045 ± 0.8522	0.672	2^{2}	2^{-3}	10^{-3}
Servo (167 × 5)	I	0.5946 ± 0.1996	0.3480 ± 0.0955	0.3203 ± 0.4328	1.0787 ± 0.6882	0.172	2^{-1}	2^{6}	–
	II	0.5954 ± 0.1813	0.3958 ± 0.0963	0.3085 ± 0.4131	1.1874 ± 0.7453	59.234	2^{2}	2^{2}	10^{-3}
	III	0.6585 ± 0.2825	0.3946 ± 0.1032	0.3245 ± 0.3214	0.9093 ± 0.5707	1.531	2^{-2}	2^{-3}	10^{-1}
Pyrim (74 × 28)	I	0.0700 ± 0.0512	0.0505 ± 0.0255	1.2075 ± 1.7877	1.5687 ± 2.2680	0.062	2^{1}	2^{5}	–
	II	0.0713 ± 0.0370	0.0523 ± 0.0213	2.6515 ± 4.6891	3.6612 ± 5.6400	5.719	2^{7}	2^{-2}	10^{-3}
	III	0.0682 ± 0.0385	0.0460 ± 0.0150	1.4674 ± 2.4072	1.6216 ± 2.4589	0.266	2^{2}	2^{0}	10^{-3}
Diabetes (43 × 3)	I	0.5472 ± 0.1489	0.4507 ± 0.1217	0.8810 ± 0.4853	0.4721 ± 0.5514	0.188	2^{-2}	2^{0}	–
	II	0.5248 ± 0.1141	0.4455 ± 0.1019	0.8827 ± 0.5119	0.7018 ± 0.8099	1.703	2^{3}	2^{-2}	10^{-3}
	III	0.5449 ± 0.1746	0.4430 ± 0.1388	0.8791 ± 0.5333	0.5034 ± 0.4918	0.078	2^{-1}	2^{-3}	10^{-3}
Triazines (186 × 61)	I	0.1350 ± 0.0229	0.0970 ± 0.0187	0.8452 ± 0.2064	0.4613 ± 0.2919	0.281	2^{1}	2^{2}	–
	II	0.1394 ± 0.0181	0.1045 ± 0.0155	0.9292 ± 0.3095	0.9073 ± 0.4395	78.500	2^{7}	2^{-3}	10^{-3}
	III	0.1375 ± 0.0233	0.0996 ± 0.0197	0.8829 ± 0.2310	0.3970 ± 0.2826	2.094	2^{2}	2^{-3}	10^{-3}
Chwirut1 (214 × 2)	I	3.1882 ± 1.6054	2.4124 ± 1.2452	0.0228 ± 0.0181	1.0314 ± 0.1378	0.250	2^{-4}	2^{7}	–
	II	3.0981 ± 1.6166	2.3206 ± 1.2609	0.0219 ± 0.0179	1.0283 ± 0.1382	74.188	2^{-1}	2^{-3}	10^{-3}
	III	3.0958 ± 1.6195	2.3702 ± 1.2678	0.0216 ± 0.0173	1.0136 ± 0.1272	3.156	2^{-4}	2^{-3}	10^{-3}
TH (49 × 2)	I	0.1423 ± 0.0889	0.1105 ± 0.0662	4.5872 ± 6.2761	6.7799 ± 8.8155	0.047	2^{-4}	2^{7}	–
	II	0.1289 ± 0.0603	0.1019 ± 0.0457	38.8481 ± 101.6	36.6093 ± 92.52	2.610	2^{-4}	2^{-1}	10^{-3}
	III	0.1156 ± 0.0870	0.0850 ± 0.0627	2.7808 ± 5.5158	3.1778 ± 4.1846	0.063	2^{-4}	2^{-3}	10^{-3}
Ozone (111 × 4)	I	16.7389 ± 3.9552	12.9165 ± 2.6387	1.0352 ± 1.0391	1.2151 ± 1.2377	0.063	2^{-2}	2^{2}	–
	II	18.4786 ± 4.8282	14.3204 ± 3.8087	1.2061 ± 0.9698	1.5112 ± 1.2367	17.750	2^{2}	2^{-2}	10^{-3}
	III	16.5982 ± 5.0727	12.5747 ± 3.4452	0.9535 ± 1.0899	1.0898 ± 1.3009	0.594	2^{-3}	2^{-3}	10^{-3}
Pollution (60 × 16)	I	39.0056 ± 9.7945	31.2016 ± 8.4370	0.5666 ± 0.2715	0.6859 ± 0.4544	0.063	2^{0}	2^{2}	–
	II	44.1408 ± 12.9426	34.0858 ± 9.2944	0.7603 ± 0.4149	1.4594 ± 0.7717	3.797	2^{7}	2^{-1}	10^{-3}
	III	31.9896 ± 4.2562	30.7073 ± 4.7026	20.6372 ± 13.3518	19.6531 ± 13.3335	0.109	2^{7}	2^{-3}	10^{-3}
Con. S (103 × 8)	I	7.6358 ± 1.7264	6.1203 ± 1.5636	1.2881 ± 0.5631	0.6725 ± 0.6836	0.203	2^{0}	2^{0}	–
	II	6.8768 ± 1.2611	5.6387 ± 1.1460	1.2121 ± 0.8423	1.1441 ± 0.9515	10.656	2^{5}	2^{3}	10^{-3}
	III	7.3816 ± 1.3638	6.0573 ± 1.3679	1.3231 ± 0.8316	0.8540 ± 0.8016	0.422	2^{1}	2^{-3}	10^{-3}
Bodyfat (252 × 15)	I	0.0043 ± 0.0031	0.0025 ± 0.0014	0.0736 ± 0.0843	0.8775 ± 0.1364	0.438	2^{0}	2^{4}	–
	II	0.0027 ± 0.0029	0.0013 ± 0.0009	0.0390 ± 0.0713	0.9677 ± 0.0920	257.938	2^{7}	2^{-3}	10^{-3}
	III	0.0023 ± 0.0022	0.0013 ± 0.0007	0.0321 ± 0.0669	0.9488 ± 0.1003	174.781	2^{7}	2^{-2}	10^{-3}

the traditional search method, but it can be very easily and efficiently implemented. In the experiments, we utilized the initial points (C_0, γ_0), $(C_0, \epsilon_0, \gamma_0)$ and $(\nu_0, \epsilon_0, \gamma_0)$ as the starting points for the LSSVR, QPTSVR, and LPTSVR, respectively. Then, we search the local optimal values. During the search process, when we find the optimal value for one parameter step by step, other parameters are fixed. From Ref. [4], we know that γ is a more important parameter compared to others. So we first fixed C (or ν) and ϵ to search the optimal value for γ. As for the initial point, we may employ some techniques [4] to estimate it, but the computational complexity is expensive. Hence, we used the fixed point, say $C_0(or\nu_0) = 2^{-3}, \epsilon_0 = 0.001, \gamma_0 = 2^{-3}$, as the initial point instead. The parameter γ was selected over the range $\{2^i | i = -4, -3, \ldots, 7\}$, C or ν was searched in the range $\{2^i | i = -4, -3, \ldots, 7\}$, and ϵ was selected from the set $\{0.001, 0.01, 0.1, 0.2\}$. The experimental results are summarized in Table 4 which consists of the mean values of RMSE, MAE, SSE/SST, SSR/SST on each dataset, the standard deviation values of the four measures on each dataset, the computation time, and the optimal parameter values.

The average ranks of the three algorithms on four measures are shown in Table 5. We can calculate that the F_F value on RMSE is 3.8011, which is larger than the critical value 3.340, so we reject the null hypothesis. It has been seen that the performance of the LPTSVR is better than those of the QPTSVR and LSSVR $(2.2667 - 1.4667 = 0.8 > 0.7493)$. Also, we obtain that the F_F value on MAE is 4.4487 and conclude that the LPTSVR performs significantly better than the QPTSVR $(2.3 - 1.4333 = 0.8667 > 0.7493)$ and the LSSVR $(2.2667 - 1.4333 = 0.8334 > 0.7493)$. The F_F value on SSE/SST is 1.9899, which is smaller than the critical value 3.340 or 2.503, so there is no significant difference among the three algorithms on SSE/SST. For SSR/SST, the F_F value is 13.8756, which is larger than 3.340, so we reject the null hypothesis. By the Nemenyi test, we can see that the QPTSVR performs best.

Besides, the LSSVR still spends on the least CPU time among the three algorithms, while the LPTSVR needs less CPU time than the QPTSVR.

Table 5 Average ranks of LSSVR, QPTSVR, and LPTSVR with RBF kernel

Algorithm	RMSE	MAE	SSE/SST	SSR/SST
LSSVR	2.2667	2.2667	1.8667	2.5333
QPTSVR	2.2667	2.3	2.4	1.2
LPTSVR	1.4667	1.4333	1.7333	2.2667

6 Conclusion

In this paper, we have proposed an approach to data regression, termed LPTSVR. In the LPTSVR, we solve twin linear programming problems instead of quadratic programming problems as one does in the QPTSVR. This makes the LPTSVR robust and faster than the QPTSVR. The LPTSVR has been extended to nonlinear regression by using nonlinear kernel techniques. The experimental results show that the generalization of the LPTSVR compares favorably with the QPTSVR and LSSVR for both linear and nonlinear cases. Besides, the LPTSVR is also suitable for weighted learning by adjusting the parameters ϵ_1 and ϵ_2. However, as does the QPTSVR [22], the LPTSVR also loses sparsity. The further work includes finding a feature selection method for the LPTSVR which is capable of generating sparse solutions.

Acknowledgments The authors gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation. The work is supported by the National Science Foundation of China (Grant No. 70601033) and Innovation Fund for Graduate Student of China Agricultural University (Grant No. KYCX2010105).

Appendix

We show the computational process of the average rank of the three algorithms on RMSE values in Table 6.

Table 6 Ranks of LSSVR, QPTSVR, and LPTSVR with linear kernel on RMSE values

Dataset	LSSVR	QPTSVR	LPTSVR
BH	2	3	1
MCPU	2	3	1
Con. CS	2	3	1
Wis. BC	2	3	1
AutoMPG	3	2	1
AutoPrice	2	3	1
Servo	3	1	2
Pyrim	2	3	1
Diabetes	2	3	1
Triazines	1	3	2
Chwirut1	3	1	2
TH	3	2	1
Mg	1.5	1.5	3
Con. S	1	3	2
Bodyfat	3	2	1
Average rank	2.167	2.433	1.4

References

1. Bi J, Bennett KP (2003) A geometric approach to support vector regression. Neurocomputing 55:79–108
2. Blake CI, Merz CJ (1998) UCI repository for machine learning databases. [http://www.ics.uci.edu/~mlearn/MLRepository.html]
3. Chang C-C, Lin C-J (2001) LIBSVM: a library for support vector machines. [http://www.csie.ntu.edu.tw/~cjlin]
4. Cherkassky V, Ma YQ (2004) Practical selection of SVM parameters and noise estimation for SVM regression. Neural Netw 17:113–126
5. Christianini V, Shawe-Taylor J (2002) An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, Cambridge
6. De Boor C, Rice JR (1968) Least-squares cubic spline approximation. II: variable knots, CSD Technical Report 21, Purdue University, IN
7. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
8. Ghorai S, Mukherjee A, Dutta PK (2009) Nonparallel plane proximal classifier. Signal Proc 89:510–522
9. Ghorai S, Hossain SJ, Mukherjee A, Dutta PK (2010) Newton's method for nonparallel plane proximal classifier with unity norm hyperplanes. Signal Proc 90:93–104
10. Jayadeva, Khemchandani R, Chandra S (2007) Twin support vector machines for pattern classification. IEEE Trans Pattern Anal Mach Intell 29(5):905–910
11. Joachims T (1999) Making large-scale SVM learning practical. In: Schölkopf B, Burges CJC, Smola AJ (eds) Advances in Kernel methods–support vector learning. MIT Press, Cambridge, pp 169–184
12. Jiao L, Bo L, Wang L (2007) Fast sparse approximation for least squares support vector machine. IEEE Trans Neural Netw 18(3):685–697
13. Keerthi SS, Shevade SK, Bhattacharyya C, Murthy K (2001) Improvements to Platt's SMO algorithm for SVM classifier design. Neural Comput 13(3):637–649
14. Keerthi SS, Shevade SK (2003) SMO algorithm for least squares SVM formulations. Neural Comput 15(2):487–507
15. Kumar MA, Gopal M (2008) Application of smoothing technique on twin support vector machines. Pattern Recogn Lett 29:1842–1848
16. Kumar MA, Gopal M (2009) Least squares twin support vector machines for pattern classification. Expert Syst Appl 36:7535–7543
17. Kruif BJ, Vries A (2004) Pruning error minimization in least squares support vector machines. IEEE Trans Neural Netw 14(3):696–702
18. Lee Y-J, Hsieh W-F, Huang C-M (2005) ε-SSVR: a smooth support vector machine forε -insensitive regression. IEEE Trans Knowl Data Eng 17(5):678–685
19. Mangasarian OL, Wild EW (2006) Multisurface proximal support vector classification via generalized eigenvalues. IEEE Trans Pattern Anal Mach Intell 28(1):69–74
20. Mangasarian OL (2006) Exact 1-norm support vector machine via unconstrained convex differentiable minimization. J Mach Learn Res 7:1517–1530
21. Osuna E, Freund R, Girosi F (1997) An improved training algorithm for support vector machines. In: Principe J, Gile L, Morgan N, Wilson E (eds) Neural networks for signal processing VII–proceedings of the 1997 IEEE workshop. IEEE. pp 276–285
22. Peng X (2009) TSVR: an efficient twin support vector machine for regression. Neural Netw. doi:10.1016/j.neunet.2009.07.002
23. Platt JC (1999) Fast training of support vector machines using sequential minimal optimization. In: Schölkopf B, Burges CJC, Smola AJ (eds) Advances in kernel methods–support vector learning. MIT Press, Cambridge, pp 185–208
24. Shevade SK, Keerthi SS, Bhattacharyya C, Murthy KRK (2000) Improvements to the SMO algorithm for SVM regression. IEEE Trans Neural Netw 11(5):1188–1193
25. Suykens JAK, Vandewalle J (1999) Least squares support vector machine classifiers. Neural Proc Lett 9(3):293–300
26. Suykens JAK, Gestel T, Brabanter J, Moor B, Vandewalle J (2002) Least squares support vector machines. World Scientific, Singapore
27. Vapnik VN (1995) The natural of statistical learning theory. Springer, New York
28. Vapnik VN (1998) Statistical learning theory. Wiley, New York
29. Wang W, Xu Z (2004) A heuristic training for support vector regression. Neurocomputing 61:259–275
30. Zeng XY, Chen XW (2005) SMO-based pruning methods for sparse least squares support vector machines. IEEE Trans Neural Netw 16(6):1541–1546
31. Zhao Y, Sun J (2009) Recursive reduced least squares support vector regression. Pattern Recogn 42:837–842