COMMUNICATIONS IN STATISTICS - SIMULATION AND COMPUTATION®
https://doi.org/10.1080/03610918.2019.1599016

Iteratively reweighted least square for asymmetric L_2-Loss support vector regression

Songfeng Zheng

Department of Mathematics, Missouri State University, Springfield, Missouri, USA

ABSTRACT
In support vector regression (SVR) model, using the squared ϵ-insensitive loss function makes the objective function of the optimization problem strictly convex and yields a more concise solution. However, the formulation leads to a quadratic programing which is expensive to solve. This paper reformulates the optimization problem by absorbing the constraints in the objective function, and the new formulation shares similarity with weighted least square regression problem. Based on this formulation, we propose an iteratively reweighted least square approach to train the L_2-loss SVR, for both linear and nonlinear models. The proposed approach is easy to implement, without requiring any additional computing package other than basic linear algebra operations. Numerical studies on real-world datasets show that, compared to the alternatives, the proposed approach can achieve similar prediction accuracy with substantially higher time efficiency.

ARTICLE HISTORY
Received 23 January 2018
Accepted 18 March 2019

KEYWORDS
Support vector regression; Squared ϵ-insensitive loss function; Weighted least square; Quadratic programing

1. Introduction

Support vector regression (SVR) model (Smola and Schölkopf 2004; Vapnik 1998) is a widely used regression technique, which is trained by minimizing the total ϵ-insensitive loss on the training set with a ridge penalty on the regression coefficients. Similar to the introduction of squared hinge loss to support vector machine (SVM) classifier (Mangasarian and Musicant 2001), squared ϵ-insensitive loss function was introduced to SVR (Balasundaram and Kapil 2010; Lee, Hsieh, and Huang 2005; Musicant and Feinberg 2004), resulting in L_2-SVR. It was shown (Balasundaram and Kapil 2010; Musicant and Feinberg 2004) that compared to the SVR model with ϵ-insensitive loss, L_2-SVR has a strictly convex objective function and more concise solution.

Same as the original SVR model, the formulation of L_2-SVR leads to a quadratic programing, which can be solved by popular SVM toolboxes (Chang and Lin 2011; Gunn 1997; Joachims 1999). Although the decomposition techniques (Osuna, Freund, and Girosi 1997a,b) or sequential minimization methods (Platt 1998) are employed to speed up the quadratic programing solver, the computational complexity is about $O(n^3)$, where n is the training set size. Thus, in general, training an SVR model is time

CONTACT Songfeng Zheng SongfengZheng@MissouriState.edu Department of Mathematics, Missouri State University, Springfield, Missouri 65897, USA.

consuming. Hence, it is highly desirable to develop a time-efficient yet accurate enough training algorithm for SVR and L_2-SVR.

In literature, there are various works (Balasundaram and Kapil 2010; Lee, Hsieh, and Huang 2005; Musicant and Feinberg 2004) to speed up the training of L_2-SVR model, and this paper gives another attempt for this purpose. Toward this end, we first reformulate the optimization problem for L_2-SVR by absorbing the constraints in a single objective function, which shares similarity with the formulation of weighted least square regression, with the weights depending on the current estimation. This observation inspires us to design an iteratively reweighted least square (IRLS) approach for training the L_2-SVR model, and the resulting algorithm is called IRLS-SVR. In this paper, we give detailed derivation process and description of the algorithm, for both linear and nonlinear L_2-SVR models. The proposed approach is easy to implement, without requiring any additional computing package other than basic linear algebra operations.

On several publicly available datasets, we compared the performances (in terms of the prediction accuracy and training time) of the proposed IRLS-SVR to several alternatives including the original quadratic programing based SVR (QP-SVR) (Gunn 1997), Lagrangian SVR (LSVR) (Balasundaram and Kapil 2010), and the smoothly approximated SVR (SSVR) (Lee, Hsieh, and Huang 2005), all using the squared ϵ-insensitive loss function. The comparison shows that IRLS-SVR can achieve similar prediction accuracy to the alternatives. However, in the training stage, IRLS-SVR is hundreds to thousands of times faster than QP-SVR, and tens to hundreds of times faster than LSVR and SSVR.

The rest of this paper is organized as follows: Sec. 2 briefly reviews the formulation of SVR with squared ϵ-insensitive loss function; Sec. 3 reformulates the L_2-SVR optimization problem as a single objective function and introduces the iteratively reweighted least square regression approach to get a solution to the model; related works are also discussed in Sec. 3; Sec. 4 compares the performance of the proposed IRLS-SVR algorithm to some alternatives on several real-world datasets, in terms of prediction error and training time; finally, Sec. 5 summarizes this paper and discusses some future research directions.

2. L_2 support vector regression

Assume the given training dataset is $\{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$, with predictor vector $\mathbf{x}_i \in \mathbb{R}^p$ and output $y_i \in \mathbb{R}$. In the case of linear regression model, we suppose the regression function is

$$y = \mathbf{w}'\mathbf{x} + b, \tag{1}$$

where $\mathbf{w} \in \mathbb{R}^p$ is the regression coefficient vector and $b \in \mathbb{R}$ is the intercept. In the case of nonlinear model, we first assume there is a map $\phi(\cdot)$ which transfers the input predictor vector $\mathbf{x}_i$ into a high dimensional space $\mathcal{H}$, and the purpose is to fit a linear model in $\mathcal{H}$, i.e.,

$$y = \mathbf{w}'\phi(\mathbf{x}) + b, \tag{2}$$

where $\mathbf{w} \in \mathcal{H}$ is the regression coefficient vector and $b \in \mathbb{R}$ is the intercept. Since the linear model is a special case of nonlinear model, we only discuss the nonlinear case in this section.

Support vector regression (SVR) employs a loss function which is not activated if the difference between the observation y_i and the prediction (i.e., $\mathbf{w}'\phi(\mathbf{x}_i)+b$) is less than a certain predefined level ϵ. This loss function is often called ϵ-insensitive loss in literature. The SVR model is obtained by solving the following constrained minimization problem

$$\begin{cases} \min_{\mathbf{w},b,\xi^*,\xi} & \frac{1}{2}\mathbf{w}'\mathbf{w}+C\left[\sum_{i=1}^{n}\xi_i+\sum_{i=1}^{n}\xi_i^*\right], \\ \text{s.t.} & y_i-\mathbf{w}'\phi(\mathbf{x}_i)-b\le\epsilon+\xi_i, \\ & \mathbf{w}'\phi(\mathbf{x}_i)+b-y_i\le\epsilon+\xi_i^*, \\ & \xi_i\ge 0 \quad\text{and}\quad \xi_i^*\ge 0,\ \text{for } i=1,2,\ldots,n. \end{cases} \tag{3}$$

In the above formulation, ϵ is the tolerance of the error, $\boldsymbol{\xi}=(\xi_1,\xi_2,\ldots,\xi_n)'$ and $\boldsymbol{\xi}^*=(\xi_1^*,\xi_2^*,\ldots,\xi_n^*)'$, with ξ_i and ξ_i^* being the nonnegative slack variables, which are the part of error which exceeds the tolerance level ϵ. As another way of understanding, ξ_i and ξ_i^* measure the effort we should pay in order to bring the prediction (i.e., $\mathbf{w}'\phi(\mathbf{x}_i)+b$) to an ϵ-neighborhood of the given observation y_i. Similar to support vector machine (SVM) classifier (Vapnik 1998), the term $\frac{1}{2}\mathbf{w}'\mathbf{w}$ in Eq. (3) measures the complexity of the regression model, and the parameter $C>0$ balances the model complexity and the model errors.

In SVM literature, it was argued (Mangasarian and Musicant 2001) that using a squared hinge loss function and penalizing on the bias will make the resulting optimization problem strictly convex and result in simpler solutions. Similarly, the squared ϵ-insensitive loss function was introduced to SVR model, and a penalty was put on the bias term (Balasundaram and Kapil 2010; Lee, Hsieh, and Huang 2005; Musicant and Feinberg 2004). Furthermore, to make the model more general, we assume that the penalty to the two situations (i.e., the prediction is above or below the observation) are different, that is, the optimization problem is

$$\begin{cases} \min_{\mathbf{w},b,\xi^*,\xi} & \frac{1}{2}\mathbf{w}'\mathbf{w}+\frac{1}{2}b^2+\frac{C}{2}\left[\sum_{i=1}^{n}w_p\xi_i^2+\sum_{i=1}^{n}w_n\left(\xi_i^*\right)^2\right] \\ \text{s.t.} & y_i-\mathbf{w}'\phi(\mathbf{x}_i)-b\le\epsilon+\xi_i \quad\text{and} \\ & \mathbf{w}'\phi(\mathbf{x}_i)+b-y_i\le\epsilon+\xi_i^*,\ \text{for } i=1,2,\ldots,n, \end{cases} \tag{4}$$

where $w_p>0$ and $w_n>0$ are the weights, and could be different. By introducing the weights w_p and w_n, Eq. (4) actually employs the asymmetric squared ϵ-insensitive loss function. Asymmetric loss functions have been considered in many regression and classification models, for example, quantile regression (Koenker 2005; Koenker and Bassett 1978), expectile regression (Efron 1991; Newey and Powell 1987), and asymmetric least square SVM (Huang, Shi, and Suykens 2014). We further notice that if we let $\epsilon=0$ in our model, we will recover the expectile regression model. Therefore, our model is a generalization to expectile regression.

By using the asymmetric squared ϵ-insensitive loss function, the objective function in Eq. (4) becomes strictly convex because all the terms in the objective function are square terms, as opposed to the linear terms in Eq. (3). We further notice that Eq. (4) does not have the non-negativity constraint on ξ_i's and ξ_i^*'s, compared to Eq. (3). Indeed, if any component of $\boldsymbol{\xi}$ is negative, say, $\xi_j<0$, we can just set $\xi_j = 0$, which makes the objective function in Eq. (4) decrease while still keeps the corresponding constraint valid. The same argument holds for $\boldsymbol{\xi}^*$. Thus, with the squared ϵ-insensitive loss function, not only is the objective function strictly convex, but also the constraints are simpler.

The Lagrangian function of the problem in Eq. (4) is

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = {} & \frac{1}{2}\mathbf{w}'\mathbf{w} + \frac{1}{2}b^2 + \frac{C}{2}\left[\sum_{i=1}^n w_p \xi_i^2 + \sum_{i=1}^n w_n \left(\xi_i^*\right)^2\right] \\ & -\sum_{i=1}^n \alpha_i\left(\epsilon + \xi_i - y_i + \mathbf{w}'\phi(\mathbf{x}_i) + b\right) - \sum_{i=1}^n \alpha_i^*\left(\epsilon + \xi_i^* + y_i - \mathbf{w}'\phi(\mathbf{x}_i) - b\right), \end{aligned}$$

where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)'$ and $\boldsymbol{\alpha}^* = (\alpha_1^*, \dots, \alpha_n^*)'$, with $\alpha_i \geq 0$ and $\alpha_i^* \geq 0$ being the Lagrangian multiplier for the i-th constraint. At the minimal point, we will have

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^n \alpha_i \phi(\mathbf{x}_i) + \sum_{i=1}^n \alpha_i^* \phi(\mathbf{x}_i) = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^n \left(\alpha_i - \alpha_i^*\right)\phi(\mathbf{x}_i), \\ \frac{\partial L}{\partial b} &= b + \sum_{i=1}^n \left(\alpha_i^* - \alpha_i\right) = 0 \Rightarrow b = \sum_{i=1}^n \left(\alpha_i - \alpha_i^*\right), \\ \frac{\partial L}{\partial \xi_i} &= C w_p \xi_i - \alpha_i = 0 \Rightarrow \xi_i = \frac{\alpha_i}{C w_p}, \end{aligned}$$

and

$$\frac{\partial L}{\partial \xi_i^*} = C w_n \xi_i^* - \alpha_i^* = 0 \Rightarrow \xi_i^* = \frac{\alpha_i^*}{C w_n}.$$

Substituting these results to the function $L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*)$, we have

$$\begin{aligned} L = {} & -\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \left(\alpha_i - \alpha_i^*\right)\phi(\mathbf{x}_i)'\phi(\mathbf{x}_j)\left(\alpha_j - \alpha_j^*\right) - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \left(\alpha_i - \alpha_i^*\right)\left(\alpha_j - \alpha_j^*\right) \\ & + \sum_{i=1}^n \left(\alpha_i - \alpha_i^*\right)y_i - \epsilon\sum_{i=1}^n \left(\alpha_i + \alpha_i^*\right) - \frac{1}{2C}\sum_{i=1}^n \frac{\alpha_i^2}{w_p} - \frac{1}{2C}\sum_{i=1}^n \frac{\left(\alpha_i^*\right)^2}{w_n}. \end{aligned} \tag{5}$$

Let $K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})'\phi(\mathbf{v})$ be the associated kernel function which satisfies Mercer's condition (Smola and Schölkopf 2004; Vapnik 1998). The dual problem is to maximize function L defined in Eq. (5), or equivalently, to minimize – L, with constraints $\alpha_i \geq 0$ and $\alpha_i^* \geq 0$ for $i = 1, 2, \dots, n$, that is

$$\begin{cases} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha}^*} & \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \left(\alpha_i - \alpha_i^*\right)K(\mathbf{x}_i, \mathbf{x}_j)\left(\alpha_j - \alpha_j^*\right) + \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \left(\alpha_i - \alpha_i^*\right)\left(\alpha_j - \alpha_j^*\right) \\ & -\sum_{i=1}^n \left(\alpha_i - \alpha_i^*\right)y_i + \epsilon\sum_{i=1}^n \left(\alpha_i + \alpha_i^*\right) + \frac{1}{2Cw_p}\sum_{i=1}^n \alpha_i^2 + \frac{1}{2Cw_n}\sum_{i=1}^n \left(\alpha_i^*\right)^2 \\ \text{s.t.} & \alpha_i \geq 0 \quad \text{and} \quad \alpha_i^* \geq 0 \text{ for } i = 1, 2, \dots, n. \end{cases} \tag{6}$$

The dual optimization problem in Eq. (6) could be solved by quadratic programing toolbox, and the final model could be obtained as

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*)\phi(\mathbf{x}_i)'\phi(\mathbf{x}) + \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*)K(\mathbf{x}_i, \mathbf{x}) + \sum_{i=1}^{n} (\alpha_i - \alpha_i^*).$$

Although the quadratic programing in Eq. (6) can be solved by decomposition methods (Osuna, Freund, and Girosi 1997a,b) or sequential minimal optimization in (Platt 1998), they are computationally expensive with complexity about $O(n^3)$. Thus, as the training set gets large, it is very expensive to train an L_2-SVR model. As such, a fast training algorithm for L_2-SVR which can achieve similar accuracy as the alternative methods is highly desired.

3. Iteratively reweighted least square for L_2-SVR

In the following discussion, we denote $V_\epsilon(x)$ as the asymmetric squared ϵ-insensitive loss function, which is defined as

$$V_\epsilon(x) = \begin{cases} w_p(x-\epsilon)^2, & \text{if } x>\epsilon, \\ 0, & \text{if } |x|<\epsilon, \\ w_n(x+\epsilon)^2, & \text{if } x<-\epsilon; \end{cases} \tag{7}$$

where w_p and w_n are defined in the same way as in Eq. (4).

3.1. Linear L_2-SVR model

We first discuss the case of linear regression model, that is, the regression function is

$$f(\mathbf{x}) = \mathbf{x}'\mathbf{w} + b.$$

To absorb the constraints in Eq. (4), we can rewrite the optimization problem in terms of $V_\epsilon(\cdot)$ as

$$\begin{aligned} \min_{\mathbf{w},b} F(\mathbf{w}, b) &= \frac{1}{2}||\mathbf{w}||^2 + \frac{1}{2}b^2 + \frac{C}{2}\sum_{i=1}^{n} V_\epsilon(y_i - f(\mathbf{x}_i)) \\ &= \frac{1}{2}||\mathbf{w}||^2 + \frac{1}{2}b^2 + \frac{C}{2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i'\mathbf{w} - b - e_i)^2 d_i, \end{aligned} \tag{8}$$

where

$$d_i = \begin{cases} w_p, & \text{if } y_i - \mathbf{x}_i'\mathbf{w} - b>\epsilon, \\ 0, & \text{if } |y_i - \mathbf{x}_i'\mathbf{w} - b| \leq \epsilon, \\ w_n, & \text{if } y_i - \mathbf{x}_i'\mathbf{w} - b< - \epsilon; \end{cases} \tag{9}$$

and

$$e_i = \begin{cases} \epsilon, & \text{if } y_i - \mathbf{x}_i'\mathbf{w} - b>\epsilon, \\ 0, & \text{if } |y_i - \mathbf{x}_i'\mathbf{w} - b| \leq \epsilon, \\ -\epsilon, & \text{if } y_i - \mathbf{x}_i'\mathbf{w} - b< - \epsilon. \end{cases} \tag{10}$$

In the matrix form, the function $F(\mathbf{w}, b)$ could be written compactly as

$$F(\mathbf{w}, b) = \frac{1}{2}\mathbf{w}'\mathbf{w} + \frac{1}{2}b^2 + \frac{C}{2}(\mathbf{y}-\mathbf{X}\mathbf{w}-b\mathbf{1}-\mathbf{e})'\mathbf{D}(\mathbf{y}-\mathbf{X}\mathbf{w}-b\mathbf{1}-\mathbf{e}), \qquad (11)$$

where $\mathbf{X}$ is the $n \times p$ data matrix with each row being a data point, $\mathbf{y} = (y_1, ..., y_n)'$, $\mathbf{e} = (e_1, ..., e_n)'$, $\mathbf{1}$ is the $n \times 1$ vector with all elements being 1, and $\mathbf{D}$ is an $n \times n$ diagonal matrix with d_i's on the diagonal.

To minimize $F(\mathbf{w}, b)$ in Eq. (11), we take gradient of $F(\mathbf{w}, b)$ with respect to $\mathbf{w}$ and set it to be zero vector, yielding

$$\frac{\partial F}{\partial \mathbf{w}} = \mathbf{w} - C\mathbf{X}'\mathbf{D}(\mathbf{y} - \mathbf{X}\mathbf{w} - b\mathbf{1} - \mathbf{e}) = \mathbf{0}_p,$$

where $\mathbf{0}_p$ is the zero vector in $\mathbb{R}^p$. The above equation is equivalent to

$$\left(\frac{\mathbf{I}}{C} + \mathbf{X}'\mathbf{D}\mathbf{X}\right)\mathbf{w} + \mathbf{X}'\mathbf{D}\mathbf{1}b = \mathbf{X}'\mathbf{D}(\mathbf{y} - \mathbf{e}), \qquad (12)$$

where $\mathbf{I}$ is identity matrix of order p. Similarly, we have

$$\frac{\partial F}{\partial b} = b - C\mathbf{1}'\mathbf{D}(\mathbf{y} - \mathbf{X}\mathbf{w} - b\mathbf{1} - \mathbf{e}) = 0,$$

or

$$\mathbf{1}'\mathbf{D}\mathbf{X}\mathbf{w} + \left(\mathbf{1}'\mathbf{D}\mathbf{1} + \frac{1}{C}\right)b = \mathbf{1}'\mathbf{D}(\mathbf{y} - \mathbf{e}). \qquad (13)$$

We write Eqs. (12) and (13) in one equation compactly as

$$\begin{pmatrix} \frac{\mathbf{I}}{C} + \mathbf{X}'\mathbf{D}\mathbf{X} & \mathbf{X}'\mathbf{D}\mathbf{1} \\ \mathbf{1}'\mathbf{D}\mathbf{X} & \mathbf{1}'\mathbf{D}\mathbf{1} + \frac{1}{C} \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix} = \begin{pmatrix} \mathbf{X}'\mathbf{D}(\mathbf{y} - \mathbf{e}) \\ \mathbf{1}'\mathbf{D}(\mathbf{y} - \mathbf{e}) \end{pmatrix} = \begin{pmatrix} \mathbf{X}' \\ \mathbf{1}' \end{pmatrix}\mathbf{D}(\mathbf{y} - \mathbf{e}). \qquad (14)$$

Unfortunately, since the matrix $\mathbf{D}$ and vector $\mathbf{e}$ both depend on the unknown quantities $\mathbf{w}$ and b, we cannot directly solve for $\mathbf{w}$ and b from Eq. (14). As such, we propose an iterative scheme as below

Algorithm 1: Iterative Procedure for Linear L_2-SVR

0. Initialization: choose a starting point $\mathbf{w}^0 \in \mathbb{R}^p$ and $b^0 \in \mathbb{R}$, set the tolerance level ε, the maximum iteration number M. Set the iteration number $t = 0$.
1. Using the current $\mathbf{w}^t$ and b^t, apply Eqs. (9) and (10) to calculate d_i and e_i, and form the matrix $\mathbf{D}$ and vector $\mathbf{e}$.
2. Update $\mathbf{w}$ and b as
$$\begin{pmatrix} \mathbf{w}^{t+1} \\ b^{t+1} \end{pmatrix} = \begin{pmatrix} \frac{\mathbf{I}}{C} + \mathbf{X}'\mathbf{D}\mathbf{X} & \mathbf{X}'\mathbf{D}\mathbf{1} \\ \mathbf{1}'\mathbf{D}\mathbf{X} & \mathbf{1}'\mathbf{D}\mathbf{1} + \frac{1}{C} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{X}' \\ \mathbf{1}' \end{pmatrix}\mathbf{D}(\mathbf{y}-\mathbf{e}).$$
3. Stop if $||\mathbf{w}^{t+1} - \mathbf{w}^t|| + |b^{t+1} - b^t| < \varepsilon$ or $t = M$; otherwise, set $t = t + 1$ and go back to step 1.

4. Return the current $\mathbf{w}$ and b.

Algorithm 1 is to minimize the function $F(\mathbf{w}, b)$ defined in Eq. (11), which is similar to the objective function for weighted least square regression with ridge penalty, except that the weight matrix $\mathbf{D}$ change in each iteration. Thus, we call Algorithm 1 as the Iteratively Reweighted Least Square L_2-SVR or IRLS-SVR for short.

3.2. Nonlinear L_2-SVR model

The idea of nonlinear SVR is to first map the predictor vector $\mathbf{x}$ by a nonlinear transformation $\phi(\mathbf{x})$ to an associated high dimensional Hilbert space $\mathcal{H}$, and then construct a linear model in $\mathcal{H}$ of the form $f(\mathbf{x}) = \mathbf{w}'\phi(\mathbf{x}) + b$. Let $K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})'\phi(\mathbf{v})$ be the associated kernel function which satisfies Mercer's condition (Smola and Schölkopf 2004; Vapnik 1998). By the representation theorem (Kimeldorf and Wahba 1970), the regression function could be written as a linear combination of kernel functions evaluated at the training examples, that is,

$$f(\mathbf{x}) = \sum_{i=1}^{n} \beta_i K(\mathbf{x}, \mathbf{x}_i) + b = \sum_{i=1}^{n} \beta_i \phi(\mathbf{x}_i)'\phi(\mathbf{x}) + b.$$

Hence, in the reproducing kernel Hilbert space, the regression coefficient vector is

$$\mathbf{w} = \sum_{i=1}^{n} \beta_i \phi(\mathbf{x}_i) \in \mathcal{H}.$$

Therefore,

$$||\mathbf{w}||^2 = \mathbf{w}'\mathbf{w} = \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i\beta_j \phi(\mathbf{x}_i)'\phi(\mathbf{x}_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i\beta_j K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\beta}'\mathbf{K}\boldsymbol{\beta},$$

where $\mathbf{K}$ is the $n \times n$ kernel matrix with $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)'$.

Similar to the linear model derived in Sec. 3.1, the optimization problem for asymmetric nonlinear L_2-SVR could be written as

$$\min_{\boldsymbol{\beta}, b} F(\boldsymbol{\beta}, b) = \frac{1}{2}\boldsymbol{\beta}'\mathbf{K}\boldsymbol{\beta} + \frac{1}{2}b^2 + \frac{C}{2}\sum_{i=1}^{n} V_\epsilon\big(y_i - f(\mathbf{x}_i)\big).$$

From the definition of $V_\epsilon(x)$, we can rewrite $F(\boldsymbol{\beta}, b)$ as

$$F(\boldsymbol{\beta}, b) = \frac{1}{2}\boldsymbol{\beta}'\mathbf{K}\boldsymbol{\beta} + \frac{1}{2}b^2 + \frac{C}{2}\sum_{i=1}^{n} (y_i - \mathbf{K}_{i\cdot}\boldsymbol{\beta} - b - e_i)^2 d_i, \tag{15}$$

where $\mathbf{K}_{i\cdot}$ is the ith row of the kernel matrix $\mathbf{K}$,

$$d_i = \begin{cases} w_p, & \text{if } y_i - \mathbf{K}_{i\cdot}\boldsymbol{\beta} - b > \epsilon, \\ 0, & \text{if } |y_i - \mathbf{K}_{i\cdot}\boldsymbol{\beta} - b| \le \epsilon, \\ w_n, & \text{if } y_i - \mathbf{K}_{i\cdot}\boldsymbol{\beta} - b < -\epsilon; \end{cases}$$

and

$$e_i = \begin{cases} \epsilon, & \text{if } y_i - \mathbf{K}_{i\cdot}\boldsymbol{\beta} - b > \epsilon, \\ 0, & \text{if } |y_i - \mathbf{K}_{i\cdot}\boldsymbol{\beta} - b| \leq \epsilon, \\ -\epsilon, & \text{if } y_i - \mathbf{K}_{i\cdot}\boldsymbol{\beta} - b < -\epsilon. \end{cases}$$

In matrix form, $F(\boldsymbol{\beta}, b)$ could be written as

$$F(\boldsymbol{\beta}, b) = \frac{1}{2}\boldsymbol{\beta}'\mathbf{K}\boldsymbol{\beta} + \frac{1}{2}b^2 + \frac{C}{2}(\mathbf{y} - \mathbf{K}\boldsymbol{\beta} - b\mathbf{1} - \mathbf{e})'\mathbf{D}(\mathbf{y} - \mathbf{K}\boldsymbol{\beta} - b\mathbf{1} - \mathbf{e}), \tag{16}$$

where $\mathbf{1}$, $\mathbf{e}$, $\mathbf{y}$, and $\mathbf{D}$ are defined in the same way as in Sec. 3.1.

To minimize $F(\boldsymbol{\beta}, b)$, we take gradient with respect to $\boldsymbol{\beta}$ and set it to be zero vector, yielding

$$\frac{\partial F}{\partial \boldsymbol{\beta}} = \mathbf{K}\boldsymbol{\beta} - C\mathbf{K}'\mathbf{D}(\mathbf{y} - \mathbf{K}\boldsymbol{\beta} - b\mathbf{1} - \mathbf{e}) = \mathbf{0}_n,$$

where $\mathbf{0}_n$ is the zero vector in $\mathbb{R}^n$. Since $\mathbf{K}$ is symmetric and invertible, we can simplify the above equation as

$$\left(\frac{\mathbf{I}}{C} + \mathbf{DK}\right)\boldsymbol{\beta} + \mathbf{D1}b = \mathbf{D}(\mathbf{y} - \mathbf{e}), \tag{17}$$

here $\mathbf{I}$ is the identify matrix of order n. Similarly, there is

$$\frac{\partial F}{\partial b} = b - C\mathbf{1}'\mathbf{D}(\mathbf{y} - \mathbf{K}\boldsymbol{\beta} - b\mathbf{1} - \mathbf{e}) = 0,$$

or

$$\mathbf{1}'\mathbf{DK}\boldsymbol{\beta} + \left(\mathbf{1}'\mathbf{D1} + \frac{\mathbf{1}}{C}\right)b = \mathbf{1}'\mathbf{D}(\mathbf{y} - \mathbf{e}). \tag{18}$$

We write Eqs. (17) and (18) compactly as

$$\begin{pmatrix} \frac{\mathbf{I}}{C} + \mathbf{DK} & \mathbf{D1} \\ \mathbf{1}'\mathbf{DK} & \mathbf{1}'\mathbf{D1} + \frac{1}{C} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta} \\ b \end{pmatrix} = \begin{pmatrix} \mathbf{D}(\mathbf{y} - \mathbf{e}) \\ \mathbf{1}'\mathbf{D}(\mathbf{y} - \mathbf{e}) \end{pmatrix} = \begin{pmatrix} \mathbf{I} \\ \mathbf{1}' \end{pmatrix}\mathbf{D}(\mathbf{y} - \mathbf{e}). \tag{19}$$

Similar to the linear model, we can develop an iteratively reweighted least square algorithm for asymmetric nonlinear L_2-SVR. The resulting algorithm is only slightly different from Algorithm 1, thus we choose not to present it in detail.

3.3. Related works

There are some works in literature which applied the iteratively reweighted least square (IRLS) strategy to SVM and related models. The most similar work to our method is the one proposed in (Pérez-Cruz, Bousoño-Calzón, and Artés-Rodríguez 2005). Both our paper and (Pérez-Cruz, Bousoño-Calzón, and Artés-Rodríguez 2005) write the objective function as a quadratic function with the coefficients depending on the quantities of interest, thus there is no direct analytic solution, and the IRLS procedure was invoked. The differences between our paper and (Pérez-Cruz, Bousoño-Calzón, and Artés-Rodríguez 2005) are

- The paper (Pérez-Cruz, Bousoño-Calzón, and Artés-Rodríguez 2005) deals with the SVM classification model and the hinge loss function is used; while in our paper, the model is for regression and the loss function is the asymmetric square of ϵ-insensitive function.
- The paper (Pérez-Cruz, Bousoño-Calzón, and Artés-Rodríguez 2005) approximates the hinge loss function by a new function that has an extra parameter K, and this will introduce extra work in model selection stage. On the contrary, our paper does not need any extra parameter.
- Our paper imposes the L_2 penalty on the bias term b, which will make the objective function strictly convex in b, while (Pérez-Cruz, Bousoño-Calzón, and Artés-Rodríguez 2005) does not impose any constraint on b.
- To write the objective function in quadratic form, (Pérez-Cruz, Bousoño-Calzón, and Artés-Rodríguez 2005) rewrites $|x|$ as $x^2/|x|$, and uses $1/|x|$ as the weight to that term. However, if x is very close to zero, it will make the weight too big. To prevent this situation from happening, (Pérez-Cruz, Bousoño-Calzón, and Artés-Rodríguez 2005) uses a quadratic function to approximate hinge loss if x is close to zero, but this introduces an extra parameter to the model. In our objective function, the weights to the error terms are specified by the user, thus they will not change in each iteration of the algorithm.

Another paper (Suykens et al. 2002) imposes weights to the errors in the least square SVM for regression problems. The error function in (Suykens et al. 2002) is the square of the prediction error, while in our paper the error function is the square of ϵ-insensitive function. Further, the weights to the errors in (Suykens et al. 2002) are determined by some weight function, according to the sign and magnitude of the errors. Several possible weight functions were compared in (De Brabanter et al. 2009). Although it is a good property to automatically determine the weights, it is not the focus of the current paper, hence we leave it as a part of future investigation.

4. Comparative studies

On two public regression datasets, we compare the performances of the proposed iteratively reweighted least square approach to several classical algorithms for L_2-SVR in literature, in terms of predictive error and training time.

4.1. Experiment setup and performance measures

The considered algorithms include using a quadratic programing to solve Eq. (6), i.e. QP-SVR, Lagrangian SVR (LSVR) (Balasundaram and Kapil 2010), and the smoothly approximated L_2-SVR (SSVR) (Lee, Hsieh, and Huang 2005).

The original LSVR (Balasundaram and Kapil 2010) only deals with symmetric squared ϵ-insensitive loss function, we generalize it to asymmetric squared ϵ-insensitive loss function as follows. Let matrix $\mathbf{H}$ be the matrix obtained by adding 1 to each element of the kernel matrix $\mathbf{K}$, that is

$$\mathbf{H} = \mathbf{K} + \mathbf{1}\mathbf{1}'.$$

Define vectors $\mathbf{r}_1 = \mathbf{y} - \epsilon\mathbf{1}$ and $\mathbf{r}_2 = -\mathbf{y} - \epsilon\mathbf{1}$, and matrices

$$\mathbf{S} = Cw_pw_n\left(\tfrac{\mathbf{I}}{C} + (w_p + w_n)\mathbf{H}\right)^{-1}\left(\frac{\mathbf{I}}{Cw_p} + \mathbf{H}\right),$$
$$\mathbf{P} = w_p\left(\tfrac{\mathbf{I}}{C} + (w_p + w_n)\mathbf{H}\right)^{-1}(\mathbf{I} + Cw_n\mathbf{H}),$$

and

$$\mathbf{R} = \mathbf{Q} = Cw_pw_n\left(\frac{\mathbf{I}}{C} + (w_p + w_n)\mathbf{H}\right)^{-1}\mathbf{H}.$$

Initialize $\mathbf{u}_1^0 \in \mathbb{R}^n$ and $\mathbf{u}_2^0 \in \mathbb{R}^n$, then iteratively update $\mathbf{u}_1$ and $\mathbf{u}_2$ according to

$$\mathbf{u}_1^{t+1} = \mathbf{P}\left(\mathbf{r}_1 + \left(\mathbf{H}(\mathbf{u}_1^t - \mathbf{u}_2^t) + \left(\frac{1}{Cw_p} - \gamma\right)\mathbf{u}_1^t - \mathbf{r}_1\right)_+\right)$$
$$+\mathbf{Q}\left(\mathbf{r}_2 + \left(-\mathbf{H}(\mathbf{u}_1^t - \mathbf{u}_2^t) + \left(\frac{1}{Cw_n} - \gamma\right)\mathbf{u}_2^t - \mathbf{r}_2\right)_+\right),$$

and

$$\mathbf{u}_2^{t+1} = \mathbf{R}\left(\mathbf{r}_1 + \left(\mathbf{H}(\mathbf{u}_1^t - \mathbf{u}_2^t) + \left(\frac{1}{Cw_p} - \gamma\right)\mathbf{u}_1^t - \mathbf{r}_1\right)_+\right)$$
$$+\mathbf{S}\left(\mathbf{r}_2 + \left(-\mathbf{H}(\mathbf{u}_1^t - \mathbf{u}_2^t) + \left(\frac{1}{Cw_n} - \gamma\right)\mathbf{u}_2^t - \mathbf{r}_2\right)_+\right),$$

with $0<\gamma<\frac{2}{\max\{w_p,w_n\}C}$, and the plus function $(x)_+ = \max(0, x)$. After convergence, denote the solutions as $\mathbf{u}_1^* = (u_{1,1}^*, u_{1,2}^*, \ldots, u_{1,n}^*)'$ and $\mathbf{u}_2^* = (u_{2,1}^*, u_{2,2}^*, \ldots, u_{2,n}^*)'$, then the finally obtained model is

$$f(\mathbf{x}) = \sum_{i=1}^{n}\left(u_{1,i}^* - u_{2,i}^*\right)K(\mathbf{x}_i, \mathbf{x}) + \sum_{i=1}^{n}\left(u_{1,i}^* - u_{2,i}^*\right).$$

In (Lee, Hsieh, and Huang 2005), the nondifferentiable function $V_\epsilon(x)$ in L_2-SVR was smoothly approximated by a differentiable function

$$p_\epsilon^2(x, \tau) = w_p(p(x - \epsilon, \tau))^2 + w_n(p(-x - \epsilon, \tau))^2,$$

where

$$p(x, \tau) = x + \frac{1}{\tau}\log\left(1 + e^{-\tau x}\right),$$

and $\tau>0$ is called the smooth parameter. The objective function of L_2-SVR optimization problem for linear model could be approximated by

$$\min_{\mathbf{w},b} \quad \Phi_{\epsilon,\tau}(\mathbf{w}, b) = \frac{1}{2}\left(||\mathbf{w}||^2 + b^2\right) + \frac{C}{2}\sum_{i=1}^{n}p_\epsilon^2\left(y_i - \mathbf{w}'\mathbf{x}_i - b, \tau\right). \tag{20}$$

For any $\tau>0, \Phi_{\epsilon,\tau}(\mathbf{w}, b)$ is twice differentiable, so it can be minimized by a Newton algorithm. With τ large enough, the solution to problem (20) will be a good

Table 1. The performance of the proposed IRLS-SVR, the quadratic programing based SVR (QP-SVR), Smooth SVR (SSVR), and Lagrangian SVR (LSVR) on the house dataset. For different training set sizes (N_{tr}), we list the mean values of the asymmetric squared ϵ-insensitive error on testing set of 100 runs, with the standard deviations listed in parentheses. The best performance is marked in **bold**.

N_{tr}	Kernel	QP-SVR	SSVR	LSVR	IRLS-SVR
100	Linear	0.6217 (0.0958)	0.5572 (0.1204)	**0.1857 (0.0801)**	0.7666 (0.1383)
	Gaussian	0.1506 (0.0687)	**0.1219 (0.0586)**	0.1512 (0.0699)	0.1506 (0.0687)
200	Linear	0.2492 (0.0880)	0.2260 (0.0851)	**0.1255 (0.0825)**	0.2493 (0.0880)
	Gaussian	0.0806 (0.0448)	**0.0638 (0.0463)**	0.0805 (0.0450)	0.0806 (0.0448)
300	Linear	0.1052 (0.0674)	**0.0991 (0.0581)**	0.1016 (0.0560)	0.1086 (0.0696)
	Gaussian	0.0599 (0.0452)	**0.0492 (0.0465)**	0.0609 (0.0456)	0.0599 (0.0452)
400	Linear	0.0854 (0.0757)	**0.0821 (0.0706)**	0.0839 (0.0697)	0.0854 (0.0757)
	Gaussian	0.0495 (0.0424)	**0.0393 (0.0443)**	0.0518 (0.0435)	0.0495 (0.0424)

approximation to the original linear L_2-SVR model. This idea can be generalized to nonlinear L_2-SVR model.

In our experiments, all the involved algorithms were implemented in MATLAB, and the core of QP-SVR was implemented based on the SVM toolbox of (Gunn 1997) with the quadratic programing solver from the C++ version of LIBSVM (Chang and Lin 2011). All the experiments were conducted on a laptop computer with Interl(R) Xeon(R) CPU with 2.00 GHz and 8 GB memory, with Windows 10 operating system and MATLAB® R2015b as the platform. During all experiments that incorporated measurement of running time, one core was used solely for the experiments, and the number of other processes running on the system was minimized.

We compared all the algorithms with linear kernel and with the Gaussian kernel

$$K(\mathbf{u}, \mathbf{v}) = \exp\left\{-\frac{||\mathbf{u}-\mathbf{v}||^2}{2\sigma^2}\right\}$$

with $\sigma = 5$. We set the parameter $C = 100$ and $\epsilon = 0.5$ in the algorithms. In LSVR, SSVR, and IRLS-SVR, the maximum number of iterations was set to be 1000, and the tolerance level was set to be $\varepsilon = 10^{-4}$. The smooth parameter in SSVR was set to be $\tau = 5$ as suggested by (Lee, Hsieh, and Huang 2005). In the experiments, we set the weight factors to be $w_p = 2$ and $w_n = 1$. The parameter setting in our experiment might not be the optimal to achieve the smallest testing error. Nevertheless, our purpose is not to achieve the optimal testing error, but to compare the performances of several L_2-SVR algorithms; therefore, the comparison is fair as long as the parameter settings are the same for all the algorithms.

For each algorithm and each of the linear and nonlinear model, we partition the whole dataset as training subset and testing subset, and use the average of asymmetric squared ϵ-insensitive error on the testing set as testing error, that is, we define

$$\text{Error} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} V_\epsilon\left(y_i - \hat{f}(\mathbf{x}_i)\right),$$

where $V_\epsilon(\cdot)$ is defined as in Eq. (7), $\hat{f}(\cdot)$ is the fitted regression function, and N_{test} is the size of the testing set.

Table 2. The average and standard deviation of the training time (in seconds) of 100 runs for different training set sizes (N_{tr}), different models, and each of the considered algorithms. The best performance is given in **bold**.

N_{tr}	Kernel	QP-SVR	SSVR	LSVR	IRLS-SVR
100	Linear	2.9706 (0.0469)	**0.0106 (0.0236)**	0.1842 (0.0331)	0.0175 (0.2194)
	Gaussian	2.9703 (0.0190)	1.5003 (0.0994)	0.1900 (0.0343)	**0.0117 (0.0256)**
200	Linear	23.5717 (0.1368)	0.0106 (0.0236)	0.3262 (0.0435)	**0.0088 (0.0518)**
	Gaussian	23.9303 (0.1061)	4.4991 (0.3198)	0.3398 (0.0554)	**0.0419 (0.0306)**
300	Linear	80.8072 (0.3055)	0.0289 (0.0443)	0.5780 (0.0736)	**0.0206 (0.1005)**
	Gaussian	82.2417 (0.3130)	10.6422 (0.5912)	0.5542 (0.0601)	**0.0838 (0.0321)**
400	Linear	196.7608 (0.4135)	0.0409 (0.0418)	1.5311 (0.0700)	**0.0113 (0.0279)**
	Gaussian	200.5756 (0.5835)	27.3947 (0.8335)	1.5239 (0.0709)	**0.2006 (0.0350)**

Table 3. The average number of iterations needed for IRLS-SVR to converge on house dataset.

N_{tr}		100	200	300	400
Kernel	Linear	122.45	10.57	18.65	13.41
	Gaussian	7.35	8.74	9.48	10.19

4.2. Boston house dataset

The Boston house dataset is available at http://lib.stat.cmu.edu/datasets/boston_corrected.txt. This dataset concerns the median house price in suburb of Boston area with 506 observations, and there are 12 non-constant continuous predictor variables. We normalize the predictor variables and the response variable to have zero mean and unit standard deviation.

For different training set sizes (100, 200, 300, and 400), we randomly partition the whole dataset into training set and testing set, and evaluate the performance of the trained model on the testing set. The partition-training-testing process is repeated 100 times for each algorithm and each model.

Table 1 reports the testing accuracy for different models and different training set sizes. We observe that for all the training set sizes, overall, SSVR has the best testing accuracies, while the other three methods roughly have testing errors on the same level, for either linear or nonlinear model. We also note that for each problem, the nonlinear model has better testing accuracies. Table 2 presents the training time of each model, and we clearly observe the speed advantage of the proposed IRLS-SVR, which is often the most time-efficient in training, with hundreds to thousands of times faster than QP-SVR and tens to hundreds of times faster than SSVR and LSVR.

To further investigate the convergence property of IRLS-SVR, for the 100 runs, we calculate the average number of iterations needed to converge for both linear and nonlinear models with different training set sizes, and the results are given in Table 3. We see that in general, the IRLS-SVR for linear model converges in 20 iterations, with only one exception; for nonlinear model with Gaussian kernel, it needs about 10 iterations to converge. For one run with linear model and training set size 200, the objective function values are (1787.53, 1036.95, 887.73, 862.63, 796.78, 795.39, 771.47, 762.24, 762.22, 762.22), calculated according to Eq. (8) or (11). Similarly, for one run with Gaussian kernel and training set size 200, the objective function values are (430.29, 276.43, 205.83, 191.00, 112.72, 111.68, 111.48, 111.48), calculated from Eq. (15) or (16). These

Table 4. Testing errors of different algorithms on "cpuSmall Prototask". See the caption of Table 1 for more information.

N_{tr}	Kernel	QP-SVR	SSVR	LSVR	IRLS-SVR
200	Linear	0.5513 (0.2770)	0.6240 (0.3100)	**0.3761 (0.3241)**	0.5487 (0.2657)
	Gaussian	**0.0415 (0.0165)**	0.0416 (0.0207)	0.0417 (0.0161)	**0.0415 (0.0165)**
400	Linear	0.2598 (0.1828)	0.2622 (0.1583)	**0.2408 (0.1281)**	0.2598 (0.1828)
	Gaussian	**0.0270 (0.0101)**	0.0285 (0.0102)	0.0283 (0.0112)	**0.0270 (0.0101)**
600	Linear	0.1421 (0.1262)	0.1509 (0.1453)	**0.1397 (0.1197)**	0.1421 (0.1262)
	Gaussian	**0.0180 (0.0054)**	0.6315 (0.0113)	0.0208 (0.0069)	**0.0180 (0.0054)**
800	Linear	0.1197 (0.0664)	0.1261 (0.0771)	**0.1188 (0.0641)**	0.1197 (0.0664)
	Gaussian	**0.0147 (0.0055)**	0.0183 (0.0067)	0.0198 (0.0083)	**0.0147 (0.0055)**

Table 5. Training time of different algorithms on "cpuSmall Prototask". See the caption of Table 2 for more information.

N_{tr}	Kernel	QP-SVR	SSVR	LSVR	IRLS-SVR
200	Linear	23.4870 (0.1831)	0.0094 (0.0224)	0.3284 (0.0485)	**0.0073 (0.0289)**
	Gaussian	23.9792 (0.1019)	4.5348 (0.3599)	0.3231 (0.0445)	**0.0372 (0.0313)**
400	Linear	195.9945 (0.7452)	0.0323 (0.0467)	1.5802 (0.0849)	**0.0056 (0.0180)**
	Gaussian	201.4255 (0.7670)	1405.0278 (6.6358)	1.5711 (0.0853)	**0.2194 (0.0391)**
600	Linear	660.6458 (3.4306)	0.0480 (0.0371)	5.1578 (0.1197)	**0.0042 (0.0155)**
	Gaussian	682.1614 (4.6342)	2291.6955 (17.4476)	5.1494 (0.1295)	**0.5444 (0.0798)**
800	Linear	1591.5236 (7.3074)	0.0705 (0.0224)	8.9169 (0.1645)	**0.0059 (0.0210)**
	Gaussian	1647.2656 (5.3049)	3834.8484 (388.1649)	8.8584 (0.1668)	**1.0664 (0.1248)**

results indicate that indeed, the objective function decreases monotonically and becomes stable at the end.

4.3. The comp-activ dataset

The comp-activ dataset is a collection of a computer system activity measures, please refer to http://www.cs.toronto.edu/~delve/data/comp-activ/compActivDetail.html for detailed description of the dataset. The dataset contains 8192 data points and 25 numerical attributes. Same as in Sec. 4.2, we normalize the predictors and response to have zero mean and unit standard deviation.

We first implement the "cpuSmall Prototask" which involves using 12 attributes to predict what fraction of a CPU's processing time is devoted to a specific mode ("user mode"). For different training set sizes (200, 400, 600, and 800), we repeat the experiment in Sec. 4.2. Table 4 gives the average testing errors along with corresponding standard deviations for each training set size. It is evident from Table 4 that for either linear or nonlinear model, IRLS-SVR and QP-SVR have essentially the same testing errors; IRLS-SVR sometimes has better performance than LSVR and sometimes has worse performance, but the testing errors are on the same level; SSVR is not stable on this problem in the sense that sometimes SSVR has very good performance while sometimes the performance is very poor. In general, we can say that IRLS-SVR has similar testing errors as state-of-the-art algorithms on this task.

Table 5 shows the training time (in seconds) of the considered algorithms on the "cpuSmall Prototask", with the best marked in **bold**. As we have observed in Sec. 4.2, IRLS-SVR is the most time efficient among the considered algorithms: it is often hundreds to thousands of times faster than QP-SVR, and tens to hundreds of times faster

Table 6. The average number of iterations needed for IRLS-SVR to converge on "cpuSmall Prototask".

N_{tr}		200	400	600	800
Kernel	Linear	8.59	7.79	7.74	7.46
	Gaussian	9.95	10.72	11.34	11.43

Table 7. Testing errors of different algorithms on "cpu Prototask". See the caption of Table 1 for more information.

N_{tr}	Kernel	QP-SVR	SSVR	LSVR	IRLS-SVR
200	Linear	0.6130 (0.0741)	0.6997 (0.0905)	**0.4104 (0.0831)**	0.6133 (0.0742)
	Gaussian	0.0764 (0.0206)	0.6319 (0.0056)	**0.0763 (0.0206)**	0.0764 (0.0206)
400	Linear	0.2330 (0.0310)	0.2593 (0.0374)	**0.2096 (0.0255)**	0.2330 (0.0310)
	Gaussian	**0.0530 (0.0091)**	0.6324 (0.0075)	0.0559 (0.0119)	**0.0530 (0.0091)**
600	Linear	0.1604 (0.1518)	0.1736 (0.0792)	**0.1498 (0.1263)**	0.1604 (0.1518)
	Gaussian	**0.0433 (0.0062)**	0.6322 (0.0084)	0.0479 (0.0102)	**0.0433 (0.0062)**
800	Linear	0.1413 (0.1163)	0.1528 (0.1402)	**0.1374 (0.1074)**	0.1413 (0.1163)
	Gaussian	**0.0377 (0.0064)**	0.6342 (0.0107)	0.0433 (0.0099)	**0.0377 (0.0064)**

Table 8. Training time of different algorithms on "cpu Prototask". See the caption of Table 2 for more information.

N_{tr}	Kernel	QP-SVR	SSVR	LSVR	IRLS-SVR
200	Linear	23.5217 (0.1881)	0.0150 (0.0268)	0.3564 (0.0482)	**0.0131 (0.0481)**
	Gaussian	23.9583 (0.0603)	359.2459 (3.0075)	0.3327 (0.0435)	**0.0408 (0.0355)**
400	Linear	196.4898 (1.5512)	0.0331 (0.0401)	1.5527 (0.0927)	**0.0063 (0.0221)**
	Gaussian	201.7123 (1.4944)	1377.9845 (13.2132)	1.5317 (0.1058)	**0.2252 (0.0463)**
600	Linear	661.3625 (3.0038)	0.0597 (0.0333)	5.1422 (0.2089)	**0.0072 (0.0225)**
	Gaussian	681.6570 (1.4053)	2295.7286 (9.1175)	5.0747 (0.0983)	**0.5344 (0.0638)**
800	Linear	1594.4356 (8.0163)	0.0883 (0.0336)	8.8941 (0.1566)	**0.0094 (0.0267)**
	Gaussian	1649.2527 (14.1445)	3899.5744 (27.3398)	8.8919 (0.1556)	**1.0731 (0.1214)**

Table 9. The average number of iterations needed for IRLS-SVR to converge on "CPU Prototask".

N_{tr}		200	400	600	800
Kernel	Linear	21.68	8.19	7.69	7.57
	Gaussian	9.95	11.02	11.45	11.80

than LSVR or SSVR. We notice that SSVR sometimes spent extremely long time in training, that is because in these situations, the program reports that "*a matrix has very small condition number*", such that the SSVR algorithm actually did not converge, and this also explains the fluctuation of the testing errors given in Table 4. This problem did not happen to LSVR or IRLS-SVR.

Table 6 lists the average number of iterations needed for IRLS-SVR to converge with different training set sizes and different models on "cpuSmall Prototask". We see that the algorithm in general converges very fast, with about 10 iterations for linear model and slightly more than 10 iterations for nonlinear model with Gaussian kernel. The evolution of objective function shows similar pattern as in Sec. 4.2, we thus choose not to list them out.

We repeat the same experiment on the "cpu Prototask" which involves using 21 predictors. Table 7 gives the prediction accuracies and Table 8 shows the average training time for different algorithms and different training set sizes, along with corresponding standard deviations. From Tables 7 and 8, we observe the same pattern as we observed from the experiments on the Boston house data and "cpuSmall prototask", that is, the proposed IRLS-SVR has similar prediction accuracy as the alternatives in literature, while it is much faster in training. Table 9 lists the average number of iterations needed for IRLS-SVR to converge with different training set sizes and different models. We see that the algorithm in general converges with about 10 iterations for linear model and slightly more than 10 iterations for nonlinear model with Gaussian kernel.

In summary, our comparisons show that the proposed IRLS-SVR achieves testing errors comparable to the state-of-the-arts, although not always the best. However, compared to alternatives, IRLS-SVR is always the most time-efficient algorithm in training. On the tested datasets, IRLS-SVR is often several hundreds to thousands of times faster than QP-SVR in training; compared to SSVR and LSVR, IRLS-SVR is often tens to hundreds of times faster for nonlinear model and several times faster for linear model.

5. Conclusions and future works

The introduction of squared ϵ-insensitive loss function to support vector regression (SVR) makes the optimization problem strictly convex with simpler constraints. However, the model formulation leads to a quadratic programing, which is computationally expensive to solve. This paper gives an attempt to train the L_2-SVR model by directly minimizing the primal form of the optimization problem. We reformulate the problem as a weighted least square regression, with the weights determined by the current estimated parameter. Based on this new formulation, an iteratively reweighted least square (IRLS) based L_2-SVR algorithm was proposed.

Experiments were conducted on several publicly available datasets, and we compared the performance of the proposed IRLS-SVR to that of the quadratic programing based SVR (QP-SVR), Lagrangian SVR (LSVR), and Smoothed SVR (SSVR), in terms of testing accuracy and the training time. Our results show that IRLS-SVR has very similar testing error as the state-of-the-arts, for either linear or nonlinear model. However, compared to alternative methods, IRLS-SVR is much faster in training. Specifically, IRLS-SVR is hundreds to thousands of times faster than QP-SVR for both linear and nonlinear models, and compared to SSVR and LSVR, IRLS-SVR is tens to hundreds of times faster for nonlinear model and several times faster for linear model.

There are several aspects of the proposed algorithm which deserve further investigation. Firstly, in this paper, we experimentally verified that the proposed IRLS-SVR algorithm converges quickly, however, there is a need to investigate the convergence property theoretically. Secondly, the proposed approach can be easily generalized to SVR models with different weights for each data point (Han and Clemmensen 2014), and it could also be generalized such that the weight for each data point is automatically determined, similar to the works in (De Brabanter et al. 2009; Suykens et al. 2002). Due to the limitation of the available computing resource, we did not test the proposed IRLS-SVR on very large datasets. Hence, thirdly, we would like to test IRLS-SVR on some large datasets as used in (Ho and Lin 2012), and compare the performance with the alternatives (Ho and Lin 2012; Mangasarian and Musicant 2002).

Acknowledgments

The author would like to extend his sincere gratitude to the anonymous reviewers for their constructive suggestions, which helped improve the quality of this paper.

Funding

This work was supported by a Faculty Research Grant from Missouri State University (F07336-162001-022).

References

Balasundaram, S. Kapil. 2010. On Lagrangian support vector regression. *Expert Systems with Applications* 37 (12):8784–92.

Chang, C. C., and C. J. Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2 (3):1–27:27. doi:10.1145/1961189.1961199.

De Brabanter, K., Pelckmans, K. De Brabanter, J. Debruyne, M. Suykens, J.A.K., Hubert, M. and B. De Moor. 2009. Robustness of kernel based regression: A comparison of iterative weighting schemes. In Proc. of the 19th International Conference on Artificial Neural Networks (ICANN), 100–110, Limassol, Cyprus.

Efron, B. 1991. Regression percentiles using asymmetric squared error loss. *Statistica Sinica* 55: 93–125.

Gunn, S. R. 1997. Support vector machines for classification and regression. *Technical report, image speech and intelligent systems research group*. Southampton, UK: University of Southampton.

Han, X., and L. Clemmensen. 2014. On weighted support vector regression. *Quality and Reliability Engineering International* 30 (6):891–903. doi:10.1002/qre.1654.

Ho, C.-H., and C. J. Lin. 2012. Large-scale linear support vector regression. *Journal of Machine Learning Research* 13:3323–48.

Huang, X., L. Shi, and J. Suykens. 2014. Asymmetric least squares support vector machine classifiers. *Computational Statistics & Data Analysis* 70:395–405. doi:10.1016/j.csda.2013.09.015.

Joachims, J. 1999. Making large-scale SVM learning practical. In *Advances in kernel methods - Support vector learning*, eds. B. Schölkopf, C. Burges and A. Smola, Cambridge, USA: MIT-Press.

Kimeldorf, G. S., and G. Wahba. 1970. A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. The Annals of Mathematical Statistics 41 (2): 495–502. doi:10.1214/aoms/1177697089.

Koenker, R. 2005. *Quantile regression*. Cambridge, UK: Cambridge University Press.

Koenker, R., and G. Bassett. 1978. Regression Quantiles. *Econometrica* 46 (1):33–50. doi:10.2307/1913643.

Lee, Y. J., W. F. Hsieh, and C. M. Huang. 2005. ϵ-SSVR: a smooth support vector machine for ϵ-insensitive regression. *IEEE Transactions on Knowledge and Data Engineering* 17 (5):678–85.

Mangasarian, O. L., and D. R. Musicant. 2001. Lagrangian support vector machines. *Journal of Machine Learning Research* 1:161–77.

Mangasarian, O. L., and D. R. Musicant. 2002. Large scale kernel regression via linear programming. *Machine Learning* 46 (1/3):255–69. doi:10.1023/A:1012422931930.

Musicant, D. R., and A. Feinberg. 2004. Active set support vector regression. *IEEE Transactions on Neural Networks* 15 (2):268–75. doi:10.1109/TNN.2004.824259.

Newey, N., and J. L. Powell. 1987. Asymmetric least squares estimation and testing. *Econometrica* 55 (4):819–47. doi:10.2307/1911031.

Osuna, E., R. Freund, and F. Girosi. 1997a. An improved training algorithm for support vector machines. In Proceedings of IEEE Workshop Neural Networks for Signal Processing (NNSP ’97), 276–85, Amelia Island, FL.

Osuna, E., R. Freund, and F. Girosi. 1997b. Training support vector machines: An application to face detection. In Proceedings of IEEE CVPR ’97, Puerto Rico.

Pérez-Cruz, F., C. Bousoño-Calzón, and A. Artés-Rodríguez. 2005. Convergence of the IRWLS procedure to the support vector machine solution. *Neural Computation* 17 (1):7–18. doi: 10.1162/0899766052530875.

Platt, J. 1998. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods - Support vector learning*, eds. B. Schöelkopf, C. Burges and A. Smola, Cambridge, USA: MIT Press.

Smola, A. J., and B. Schölkopf. 2004. A tutorial on support vector regression. *Statistics and Computing* 14 (3):199–222. doi:10.1023/B:STCO.0000035301.49549.88.

Suykens, J. A. K., J. De Brabanter, L. Lukas, and J. Vandewalle. 2002. Weighted least squares support vector machines: robustness and sparse approximation. *Neurocomputing* 48 (1–4):85–105. doi:10.1016/S0925-2312(01)00644-0.

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