Large-scale Linear Support Vector Regression

Chia-Hua Ho B95082@CSIE.NTU.EDU.TW
Department of Computer Science, National Taiwan University, Taipei 106, Taiwan

Chih-Jen Lin CJLIN@CSIE.NTU.EDU.TW
Department of Computer Science, National Taiwan University, Taipei 106, Taiwan

Abstract

Support vector regression (SVR) and support vector classification (SVC) are popular learning techniques, but their use with kernels is often time consuming. Recently, linear SVC without kernels has been shown to give competitive accuracy for some applications, but enjoys much faster training/testing. However, few studies have focused on linear SVR. In this paper, we extend state-of-the-art training methods for linear SVC to linear SVR. We show that the extension is straightforward for some methods, but is not trivial for some others. Our experiments demonstrate that for some problems, the proposed linear-SVR training methods can very efficiently produce models that are as good as kernel SVR.

1 Introduction

Support vector regression (SVR) is a widely used regression technique (Vapnik, 1995). It is extended from support vector classification (SVC) by Boser et al. (1992). Both SVR and SVC are often used with the kernel trick (Cortes and Vapnik, 1995), which maps data to a higher dimensional space and employs a kernel function. We refer to such settings as nonlinear SVR and SVC. Although effective training methods have been proposed (e.g., Joachims, 1998; Platt, 1998; Chang and Lin, 2011), it is well known that training/testing large-scale nonlinear SVC and SVR is time consuming.

Recently, for some applications such as document classification, linear SVC without using kernels have shown to give competitive performances, but training and testing are much faster. A series of studies (e.g., Keerthi and DeCoste, 2005; Joachims, 2006; Shalev-Shwartz et al., 2007; Hsieh et al., 2008) have made linear classifiers (SVC and logistic regression) an effective and efficient tool. On the basis of this success, we are interested in whether linear SVR can be useful for some large-scale applications. Some available document data come with real-valued labels, so for them SVR rather than SVC must be considered. In this paper, we develop efficient training methods to demonstrate that, similar to SVC, linear SVR can sometimes achieve comparable performance to nonlinear SVR, but enjoys much faster training/testing.

We focus on methods in the popular package LIBLINEAR (Fan et al., 2008), which currently provides two types of methods for large-scale linear SVC.[1] The first is a

[1]We mean standard SVC using L2 regularization. For L1-regularized problems, the solvers are different.

Newton-type method to solve the primal-form of SVC (Lin et al., 2008), while the second is a coordinate descent approach for the dual form (Hsieh et al., 2008). We show that it is straightforward to extend the Newton method for linear SVR, but some careful redesign is essential for applying coordinate descent methods.

LIBLINEAR offers two types of training methods for linear SVC because they complement each other. A coordinate descent method quickly gives an approximate solution, but may converge slowly in the end. In contrast, Newton methods have the opposite behavior. We demonstrate that similar properties still hold when these training methods are applied to linear SVR.

This paper is organized as follows. In Section 2, we give the formulation of linear SVR and discuss some differences between SVC and SVR. In Section 3, we investigate two types of optimization methods for training large-scale linear SVR. In particular, we propose a condensed implementation of coordinate descent methods. We conduct experiments in Section 4 on some large regression problems. A comparison between linear and nonlinear SVR is given, followed by detailed experiments of optimization methods for linear SVR. Section 5 concludes this work.

2 Linear Support Vector Regression

Given a set of training instance-target pairs $\{(\boldsymbol{x}_i, y_i)\}$, $\boldsymbol{x}_i \in \mathbf{R}^n$, $y_i \in \mathbf{R}$, $i = 1, \ldots, l$, linear SVR finds a model $\boldsymbol{w}$ such that $\boldsymbol{w}^T\boldsymbol{x}_i$ is close to the target value y_i. It solves the following regularized optimization problem.

$$\min_{\boldsymbol{w}} \quad f(\boldsymbol{w}) \tag{1}$$

where

$$f(\boldsymbol{w}) \equiv \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_\epsilon(\boldsymbol{w}; \boldsymbol{x}_i, y_i),$$

$C > 0$ is the regularization parameter, and

$$\xi_\epsilon(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = \begin{cases} \max(|\boldsymbol{w}^T\boldsymbol{x}_i - y_i| - \epsilon, 0) & \text{or} \quad (2) \\ \max(|\boldsymbol{w}^T\boldsymbol{x}_i - y_i| - \epsilon, 0)^2 & (3) \end{cases}$$

is the ϵ-insensitive loss function associated with $(\boldsymbol{x}_i, y_i)$. The parameter ϵ is given so that the loss is zero if $|\boldsymbol{w}^T\boldsymbol{x}_i - y_i| \le \epsilon$. We refer to SVR using (2) and (3) as L1-loss and L2-loss SVR, respectively. It is known that L1 loss is not differentiable, while L2 loss is differentiable but not twice differentiable. An illustration of the two loss functions is in Figure 1. Once problem (1) is minimized, the prediction function is $\boldsymbol{w}^T\boldsymbol{x}$.

Standard SVC and SVR involve a bias term b so that the prediction function is $\boldsymbol{w}^T\boldsymbol{x} + b$. Recent works on large-scale linear classification often omit the bias term because it hardly affects the performance on most data. We omit a bias term b in problem (1) as well, although in Section 4.5 we briefly investigate the performance with/without it.

It is well known that the dual problem of L1-/L2-loss SVR is

$$\min_{\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-} f_A(\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-) \qquad \text{subject to} \quad 0 \le \alpha_i^+, \alpha_i^- \le U, \forall i = 1, \ldots, l, \tag{4}$$

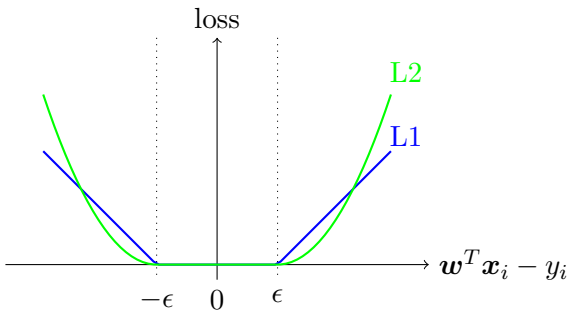


Figure 1: L1-loss and L2-loss functions.

where

$$f_A(\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-)$$
$$= \frac{1}{2}(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-)^T Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-) + \sum_{i=1}^{l} \left(\epsilon(\alpha_i^+ + \alpha_i^-) - y_i(\alpha_i^+ - \alpha_i^-) + \frac{\lambda}{2}((\alpha_i^+)^2 + (\alpha_i^-)^2)\right). \quad (5)$$

In Equation (5), $Q \in \mathbf{R}^{l\times l}$ is a matrix with $Q_{ij} \equiv \boldsymbol{x}_i^T \boldsymbol{x}_j$,

$$\lambda = \begin{cases} 0 \\ \frac{1}{2C} \end{cases}, \text{ and } U = \begin{cases} C & \text{if L1-loss SVR,} \\ \infty & \text{if L2-loss SVR.} \end{cases}$$

We can combine $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$ so that

$$\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}^+ \\ \boldsymbol{\alpha}^- \end{bmatrix} \quad \text{and} \quad f_A(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \begin{bmatrix} \bar{Q} & -Q \\ -Q & \bar{Q} \end{bmatrix} \boldsymbol{\alpha} + \begin{bmatrix} \epsilon\boldsymbol{e} - \boldsymbol{y} \\ \epsilon\boldsymbol{e} + \boldsymbol{y} \end{bmatrix}^T \boldsymbol{\alpha}, \quad (6)$$

where $\bar{Q} = Q + \lambda\mathcal{I}$, $\mathcal{I}$ is the identity matrix, and $\boldsymbol{e}$ is the vector of ones. In this paper, we refer to (1) as the primal SVR problem, while (4) as the dual SVR problem. The primal-dual relationship indicates that primal optimal solution $\boldsymbol{w}^*$ and dual optimal solution $(\boldsymbol{\alpha}^+)^*$ and $(\boldsymbol{\alpha}^-)^*$ satisfy

$$\boldsymbol{w}^* = \sum_{i=1}^{l}((\alpha_i^+)^* - (\alpha_i^-)^*)\boldsymbol{x}_i.$$

An important property of the dual problem (4) is that at optimum,

$$(\alpha_i^+)^*(\alpha_i^-)^* = 0, \forall i.$$

This result can be easily proved by seeing that the function value in (5) can become smaller if both α_i^+ and α_i^- are subtracted by a positive constant.

2.1 Differences Between SVC and SVR

SVR is very similar to SVC, although they differ in several aspects. These differences are not specific to the linear case, but we list them here to help our subsequent discussion.

1. **Labels versus target values:** SVC considers class label $y \in \{+1, -1\}$ rather than a real number.

2. **Loss functions:** The loss function of SVC is

$$\max(0, 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i) \quad \text{or} \quad \max(0, 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i)^2.$$

   In classification, we hope $y\boldsymbol{w}^T\boldsymbol{x} \geq 1$, but in regression we would like to have

$$-\epsilon \leq \boldsymbol{w}^T\boldsymbol{x} - y \leq \epsilon.$$

   Consequently, SVR has one more parameter ϵ than SVC. Parameter selection for SVR is thus more time consuming.

3. **Number of dual variables:** The dual problem of SVR has $2l$ variables, while SVC has only l. If a dual-based solver is applied without a careful design, the cost may be significantly higher than that for SVC.

3 Optimization Methods for Training Linear SVR

In this section, we extend two linear-SVC methods in LIBLINEAR for linear SVR. The first is a Newton method for L2-loss SVR, while the second is a coordinate descent method for L1-/L2-loss SVR.

3.1 A Trust Region Newton Method (TRON) for L2-loss SVR

TRON (Lin and Moré, 1999) is a general optimization method for differentiable unconstrained and bound-constrained problems, where the primal problem of L2-loss SVR is a case. Lin et al. (2008) investigate the use of TRON for L2-loss SVC and logistic regression. In this section, we discuss how TRON can be applied to solve large linear L2-loss SVR.

The optimization procedure of TRON involves two layers of iterations. At the k-th outer-layer iteration, given the current position $\boldsymbol{w}^k$, TRON sets a trust-region size Δ_k and constructs a quadratic model

$$q_k(\boldsymbol{s}) \equiv \nabla f(\boldsymbol{w}^k)^T \boldsymbol{s} + \frac{1}{2} \boldsymbol{s}^T \nabla^2 f(\boldsymbol{w}^k) \boldsymbol{s}$$

as the approximation to $f(\boldsymbol{w}^k + \boldsymbol{s}) - f(\boldsymbol{w}^k)$. Then, in the inner layer, TRON solves the following problem to find a Newton direction under a step-size constraint.

$$\min_{\boldsymbol{s}} \quad q_k(\boldsymbol{s}) \qquad \text{subject to} \quad \|\boldsymbol{s}\| \leq \Delta_k. \tag{7}$$

TRON adjusts the trust region Δ_k according to the approximate function reduction $q_k(\boldsymbol{s})$ and the real function decrease; see details in Lin et al. (2008).

To compute a truncated Newton direction by solving (7), TRON needs the gradient $\nabla f(\boldsymbol{w})$ and Hessian $\nabla^2 f(\boldsymbol{w})$. The gradient of L2-loss SVR is

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + 2C(X_{I_1,:})^T(X_{I_1,:}\boldsymbol{w} - \boldsymbol{y}_{I_1} - \epsilon \boldsymbol{e}_{I_1}) - 2C(X_{I_2,:})^T(-X_{I_2,:}\boldsymbol{w} + \boldsymbol{y}_{I_2} - \epsilon \boldsymbol{e}_{I_2}),$$

Algorithm 1 A trust region Newton method for L2-loss SVR

1. Given $\boldsymbol{w}^0$.
2. For $k = 0, 1, 2, \ldots$
   2.1. If (8) is satisfied,
        return $\boldsymbol{w}^k$.
   2.2. Solve subproblem (7).
   2.3. Update $\boldsymbol{w}^k$ and Δ_k to $\boldsymbol{w}^{k+1}$ and Δ_{k+1}.

where

$$X \equiv [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l]^T, I_1 \equiv \{i \mid \boldsymbol{w}^T \boldsymbol{x}_i - y_i > \epsilon\}, \text{ and } I_2 \equiv \{i \mid \boldsymbol{w}^T \boldsymbol{x}_i - y_i < -\epsilon\}.$$

However, $\nabla^2 f(\boldsymbol{w})$ does not exist because L2-loss SVR is not twice differentiable. Following Mangasarian (2002) and Lin et al. (2008), we use the generalized Hessian matrix. Let

$$I \equiv I_1 \cup I_2.$$

The generalized Hessian can be defined as

$$\nabla^2 f(\boldsymbol{w}) = \mathcal{I} + 2C(X_{I,:})^T D_{I,I} X_{I,:},$$

where $\mathcal{I}$ is the identity matrix, and D is an l-by-l diagonal matrix with

$$D_{ii} \equiv \begin{cases} 1 & \text{if } i \in I, \\ 0 & \text{if } i \notin I. \end{cases}$$

For large-scale problems, we can not store an n-by-n Hessian matrix in the memory. The same problem has occurred in classification, so Lin et al. (2008) applied an iterative method to solve (7). In each inner iteration, only some Hessian-vector products are required and they can be performed without storing Hessian. We consider the same setting so that for any vector $\boldsymbol{v} \in \mathbf{R}^n$,

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2C(X_{I,:})^T(D_{I,I}(X_{I,:}\boldsymbol{v})).$$

For the stopping condition, we follow the current setting in LIBLINEAR to check if the gradient is small enough in compared to the initial gradient.

$$\|\nabla f(\boldsymbol{w}^k)\|_2 \le \epsilon_s \|\nabla f(\boldsymbol{w}^0)\|_2, \tag{8}$$

where $\boldsymbol{w}^0$ is the initial iterate and ϵ_s is stopping tolerance given by users. Algorithm 1 gives the basic framework of TRON.

Similar to the situation in classification, the most expensive operation is the Hessian-vector product. It costs $O(|I|n)$ to evaluate $\nabla^2 f(\boldsymbol{w})\boldsymbol{v}$.

3.2 Dual Coordinate Descent Methods (DCD)

In this section, we introduce DCD, a coordinate descent method for the dual form of SVC/SVR. It is used in LIBLINEAR for both L1- and L2-loss SVC. We first extend the setting of Hsieh et al. (2008) to SVR and then propose a better algorithm using properties of SVR.

3.2.1 A Direct Extension from Classification to Regression

A coordinate descent method sequentially updates one variable by solving the following subproblem.

$$\begin{aligned} \min_{z} \quad & f_A(\boldsymbol{\alpha} + z\boldsymbol{e}_i) - f_A(\boldsymbol{\alpha}) \\ \text{subject to} \quad & 0 \le \alpha_i + z \le U. \end{aligned}$$

where

$$f_A(\boldsymbol{\alpha} + z\boldsymbol{e}_i) - f_A(\boldsymbol{\alpha}) = \nabla_i f_A(\boldsymbol{\alpha}) z + \frac{1}{2}\nabla^2_{ii} f_A(\boldsymbol{\alpha}) z^2$$

and $\boldsymbol{e}_i \in \mathbf{R}^{2l\times 1}$ is a vector with i-th element one and others zero. The optimal value z can be solved in a closed form, so α_i is updated by

$$\alpha_i \leftarrow \min\left(\max\left(\alpha_i - \frac{\nabla_i f_A(\boldsymbol{\alpha})}{\nabla^2_{ii} f_A(\boldsymbol{\alpha})}, 0\right), U\right), \tag{9}$$

where

$$\nabla_i f_A(\boldsymbol{\alpha}) = \begin{cases} (Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-))_i + \epsilon - y_i + \lambda\alpha_i^+, & \text{if } 1 \le i \le l, \\ -(Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-))_{i-l} + \epsilon + y_{i-l} + \lambda\alpha_{i-l}^-, & \text{if } l+1 \le i \le 2l, \end{cases}$$

and

$$\nabla^2_{ii} f_A(\boldsymbol{\alpha}) = \begin{cases} \bar{Q}_{ii} & \text{if } 1 \le i \le l, \\ \bar{Q}_{i-l,i-l} & \text{if } l+1 \le i \le 2l. \end{cases}$$

To efficiently implement (9), techniques that have been employed for SVC can be applied. First, we precalculate $\bar{Q}_{ii} = \boldsymbol{x}_i^T\boldsymbol{x}_i + \lambda, \forall i$ in the beginning. Second, $(Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-))_i$ is obtained using a vector $\boldsymbol{u}$.

$$(Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-))_i = \boldsymbol{u}^T\boldsymbol{x}_i, \text{ where } \boldsymbol{u} \equiv \sum_{i=1}^{l}(\alpha_i^+ - \alpha_i^-)\boldsymbol{x}_i.$$

If the current iterate α_i is updated to $\bar{\alpha}_i$ by (9), then vector $\boldsymbol{u}$ can be maintained by

$$\boldsymbol{u} \leftarrow \begin{cases} \boldsymbol{u} + (\bar{\alpha}_i - \alpha_i)\boldsymbol{x}_i, & \text{if } 1 \le i \le l, \\ \boldsymbol{u} - (\bar{\alpha}_{i-l} - \alpha_{i-l})\boldsymbol{x}_{i-l}, & \text{if } l+1 \le i \le 2l. \end{cases} \tag{10}$$

Both (9) and (10) cost $O(n)$, which is the same as the cost in classification.

Hsieh et al. (2008) check the projected gradient $\nabla^P f_A(\boldsymbol{\alpha})$ for the stopping condition because $\boldsymbol{\alpha}$ is optimal if and only if $\nabla^P f_A(\boldsymbol{\alpha})$ is zero. The projected gradient is defined as

$$\nabla^P_i f_A(\boldsymbol{\alpha}) \equiv \begin{cases} \min(\nabla_i f_A(\boldsymbol{\alpha}),\ 0) & \text{if } \alpha_i = 0, \\ \max(\nabla_i f_A(\boldsymbol{\alpha}),\ 0) & \text{if } \alpha_i = U, \\ \nabla_i f_A(\boldsymbol{\alpha}) & \text{if } 0 < \alpha_i < U. \end{cases} \tag{11}$$

If $\nabla^P_i f_A(\boldsymbol{\alpha}) = 0$, then (9) and (11) imply that α_i needs not be updated. We show the overall procedure in Algorithm 2.

Hsieh et al. (2008) apply two techniques to make a coordinate descent method faster. The first one is to permute all variables at each iteration to decide the order for update.

Algorithm 2 A DCD method for linear L1-/L2-loss SVR

1. Given $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$. Let $\boldsymbol{\alpha}=\begin{bmatrix}\boldsymbol{\alpha}^+\\ \boldsymbol{\alpha}^-\end{bmatrix}$ and the corresponding $\boldsymbol{u}=\sum_{i=1}^{l}(\alpha_i-\alpha_{i+l})\boldsymbol{x}_i$.
2. Compute the Hessian diagonal $\bar{Q}_{ii},\ \forall i=1,\ldots,l$.
3. For $k=0,1,2,\ldots$
   - For $i\in\{1,\ldots,2l\}$ // select an index to update
     3.1. If $|\nabla_i^P f_A(\boldsymbol{\alpha})|\neq 0$
       3.1.1. Update α_i by (9), where $(Q(\boldsymbol{\alpha}^+-\boldsymbol{\alpha}^-))_i$ is evaluated by $\boldsymbol{u}^T\boldsymbol{x}_i$.
       3.1.2. Update $\boldsymbol{u}$ by (10).

We find that this setting is also useful for SVR. The second implementation technique is shrinking. By gradually removing some variables, smaller optimization problems are solved to save the training time. In Hsieh et al. (2008), they remove those which are likely to be bounded (i.e., 0 or U) at optimum. Their shrinking strategy can be directly applied here, so we omit details.

While we have directly applied a coordinate descent method to solve (4), the procedure does not take SVR's special structure into account. Note that $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$ in (5) are very related. We can see that in the following situations some operations in Algorithm 2 are redundant.

1. We pointed out in Section 2 that an optimal $\boldsymbol{\alpha}$ of (4) satisfies

$$\alpha_i^+\alpha_i^-=0,\forall i. \tag{12}$$

   If one of α^+ or α^- is positive at optimum, it is very possible that the other is zero throughout all final iterations. Because we sequentially select variables for update, these zero variables, even if not updated in steps 3.1.1–3.1.2 of Algorithm 2, still need to be checked in the beginning of step 3.1. Therefore, some operations are wasted. Shrinking can partially solve this problem, but alternatively we may explicitly use the property (12) in designing the coordinate descent algorithm.

2. We show that some operations in calculating the projected gradient in (11) are wasted if all we need is the largest component of the projected gradient. Assume $\alpha_i^+>0$ and $\alpha_i^-=0$. If the optimality condition at α_i^- is not satisfied yet, then

$$\nabla_{i+l}^P f_A(\boldsymbol{\alpha})=\nabla_{i+l}f_A(\boldsymbol{\alpha})=-(Q(\boldsymbol{\alpha}^+-\boldsymbol{\alpha}^-))_i+\epsilon+y_i+\lambda\alpha_i^-<0.$$

   We then have

$$\begin{aligned}0<-\nabla_{i+l}f_A(\boldsymbol{\alpha})&=(Q(\boldsymbol{\alpha}^+-\boldsymbol{\alpha}^-))_i-\epsilon-y_i-\lambda\alpha_i^-\\&<(Q(\boldsymbol{\alpha}^+-\boldsymbol{\alpha}^-))_i+\epsilon-y_i+\lambda\alpha_i^+=\nabla_i f_A(\boldsymbol{\alpha}),\end{aligned} \tag{13}$$

   so a larger violation of the optimality condition occurs at α_i^+. Thus, when $\alpha_i^+>0$ and $\alpha_i^-=0$, checking $\nabla_{i+l}f_A(\boldsymbol{\alpha})$ is not necessary if we aim to find the largest element of the projected gradient.

In the next section, we propose a better coordinate descent method for SVR by combining $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$ to a vector $\boldsymbol{\alpha}^+-\boldsymbol{\alpha}^-$.

3.2.2 A New Coordinate Descent Method by Solving $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$ Together

Using the property (12), the following problem replaces $(\alpha_i^+)^2 + (\alpha_i^-)^2$ in (5) with $(\alpha_i^+ - \alpha_i^-)^2$ and gives the same optimal solutions as the dual problem (4).

$$\min_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-} \frac{1}{2}(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-)^T Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-) + \sum_{i=1}^{l} \big(\epsilon(\alpha_i^+ + \alpha_i^-) - y_i(\alpha_i^+ - \alpha_i^-) + \frac{1}{2}(\alpha_i^+ - \alpha_i^-)^2\big). \quad (14)$$

Further, Equation (12) and $\alpha_i^+ \geq 0, \alpha_i^- \geq 0$ imply that at optimum,

$$\alpha_i^+ + \alpha_i^- = |\alpha_i^+ - \alpha_i^-|.$$

With $\bar{Q} = Q + \lambda \mathcal{I}$ and defining

$$\boldsymbol{\beta} = \boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-,$$

problem (14) can be transformed as

$$\min_{\boldsymbol{\beta}} \quad f_B(\boldsymbol{\beta}) \quad \text{subject to} \quad -U \leq \beta_i \leq U, \forall i, \quad (15)$$

where

$$f_B(\boldsymbol{\beta}) \equiv \frac{1}{2}\boldsymbol{\beta}^T \bar{Q} \boldsymbol{\beta} - \boldsymbol{y}^T \boldsymbol{\beta} + \epsilon \|\boldsymbol{\beta}\|_1.$$

If $\boldsymbol{\beta}^*$ is an optimum of (15), then

$$(\alpha_i^+)^* \equiv \max(\beta_i^*, 0) \quad \text{and} \quad (\alpha_i^-)^* \equiv \max(-\beta_i^*, 0)$$

are optimal for (4).

We design a coordinate descent method to solve (15). Interestingly, (15) is in a form similar to the primal optimization problem of L1-regularized regression and classification. In LIBLINEAR, a coordinate descent solver is provided for L1-regularized L2-loss SVC (Yuan et al., 2010). We will adapt some of its implementation techniques here. A difference between L1-regularized classification and the problem (15) is that (15) has additional bounded constraints.

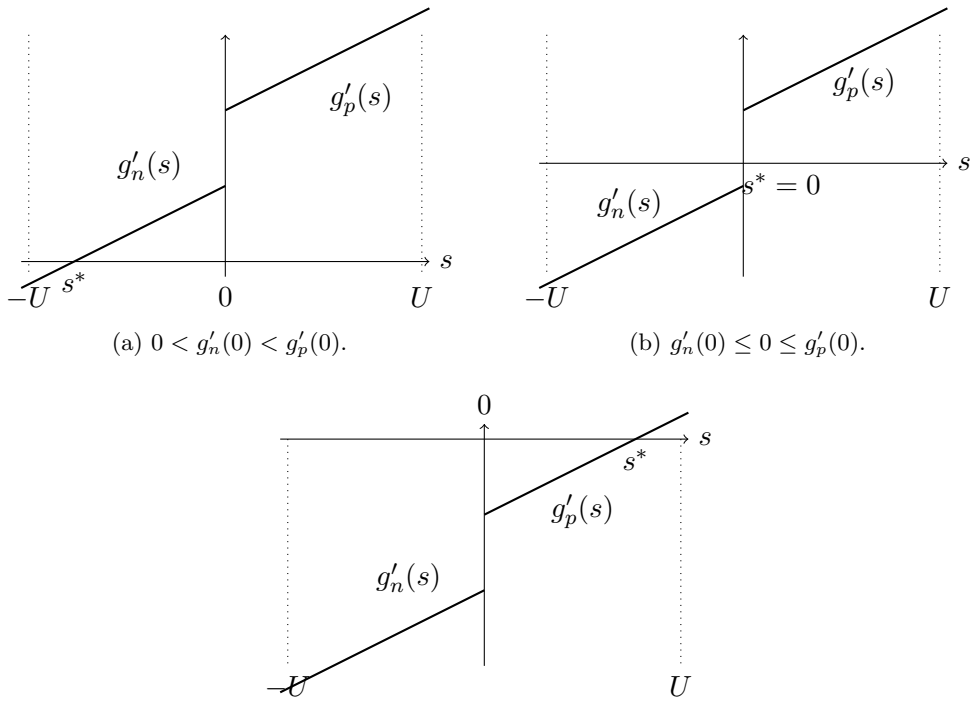
Assume $\boldsymbol{\beta}$ is the current iterate and its i-th component, denoted as a scalar variable s, is being updated. That is, $\boldsymbol{\beta}$ is a constant vector in the subsequent discussion. Then the following one-variable subproblem is solved.

$$\min_{s} \quad g(s) \quad \text{subject to} \quad -U \leq s \leq U, \quad (16)$$

where

$$\begin{aligned} g(s) &= f_B(\boldsymbol{\beta} + (s - \beta_i)\boldsymbol{e}_i) - f_B(\boldsymbol{\beta}) \\ &= \epsilon|s| + (\bar{Q}\boldsymbol{\beta} - \boldsymbol{y})_i(s - \beta_i) + \frac{1}{2}\bar{Q}_{ii}(s - \beta_i)^2 + \text{ constant}. \end{aligned} \quad (17)$$

It is well known that (17) can be reduced to "soft-thresholding" in signal processing and has a closed-form minimum. However, here we decide to give detailed derivations of solving (16) because of several reasons. First, s is now bounded in $[-U, U]$. Second, the discussion will help to explain our stopping condition and shrinking procedure.



(a) $0 < g'_n(0) < g'_p(0)$.

(b) $g'_n(0) \leq 0 \leq g'_p(0)$.

(c) $g'_n(0) < g'_p(0) < 0$.

Figure 2: We discuss the minimization of (16) using three cases. The y-axis indicates the value of $g'_p(s)$ and $g'_n(s)$. The point s^* denotes the optimal solution.

To solve (16), we start with checking the derivative of $g(s)$. Although $g(s)$ is not differentiable at $s = 0$, its derivatives at $s \geq 0$ and $s \leq 0$ are respectively

$$\begin{aligned} g'_p(s) &= \epsilon + (\bar{Q}\boldsymbol{\beta} - \boldsymbol{y})_i + \bar{Q}_{ii}(s - \beta_i) \quad \text{if } s \geq 0, \text{ and} \\ g'_n(s) &= -\epsilon + (\bar{Q}\boldsymbol{\beta} - \boldsymbol{y})_i + \bar{Q}_{ii}(s - \beta_i) \quad \text{if } s \leq 0. \end{aligned}$$

Both $g'_p(s)$ and $g'_n(s)$ are linear functions of s. Further,

$$g'_n(s) \leq g'_p(s), \forall s \in \mathcal{R}.$$

For any strictly convex quadratic function, the unique minimum occurs when the first derivative is zero. Because $g(s)$ is only piece-wise quadratic, we consider three cases in Figure 2 according to the values of $g'_p(s)$ and $g'_n(s)$. In Figure 2(a), $0 < g'_n(0) < g'_p(0)$, so $g(0)$ is the smallest on the positive side:

$$g(0) \leq g(s), \forall s \geq 0. \tag{18}$$

For $s \leq 0$, $g'_n(s) = 0$ has a root because the line of $g'_n(s)$ intersects the x-axis. With (18), this root is the minimum for both $s \leq 0$ and $s \geq 0$. By solving $g'_n(s) = 0$ and taking the condition $0 < g'_n(0)$, the solution of (16) is

$$\beta_i - \frac{-\epsilon + (\bar{Q}\boldsymbol{\beta} - \boldsymbol{y})_i}{\bar{Q}_{ii}} \quad \text{if } -\epsilon + (\bar{Q}\boldsymbol{\beta} - \boldsymbol{y})_i > \bar{Q}_{ii}\beta_i. \tag{19}$$

Algorithm 3 A new DCD method which solves (15) for linear L1-/L2-loss SVR

1. Given $\boldsymbol{\beta}$ and the corresponding $\boldsymbol{u}=\sum_{i=1}^{l}\beta_i\boldsymbol{x}_i$.
2. Compute the Hessian diagonal $\bar{Q}_{ii},\ \forall i=1,\ldots,l$.
3. For $k=0,1,2,\ldots$
   - For $i\in\{1,\ldots,l\}$ // select an index to update
     3.1. Find s by (22), where $(Q\boldsymbol{\beta})_i$ is evaluated by $\boldsymbol{u}^T\boldsymbol{x}_i$.
     3.2. $\boldsymbol{u}\leftarrow\boldsymbol{u}+(s-\beta_i)\boldsymbol{x}_i$.
     3.3. $\beta_i\leftarrow s$.

We also need to take the constraint $s\in[-U,U]$ in Equation (16) into account. If the value obtained in (19) is smaller than $-U$, then $g_n'(s)>0,\forall s\geq -U$. That is, $g(s)$ is an increasing function and the minimum is at $s=-U$.

The situation is similar in Figure 2(c), where the minimum occurs at $g_p'(s)=0$. For the remaining case in Figure 2(b),

$$g_n'(0)\leq 0\leq g_p'(0). \tag{20}$$

Inequalities in (20) imply that $g(s)$ is a decreasing function at $s\leq 0$, but is an increasing function at $s\geq 0$. Thus, an optimal solution occurs at $s=0$. A summary of the three cases shows that the subproblem (16) has the following closed form solution.

$$s\leftarrow\max(-U,\min(U,\beta_i+d)), \tag{21}$$

where

$$d\equiv\begin{cases}-\frac{g_p'(\beta_i)}{\bar{Q}_{ii}} & \text{if } g_p'(\beta_i)<\bar{Q}_{ii}\beta_i,\\ -\frac{g_n'(\beta_i)}{\bar{Q}_{ii}} & \text{if } g_n'(\beta_i)>\bar{Q}_{ii}\beta_i,\\ -\beta_i & \text{otherwise.}\end{cases} \tag{22}$$

In (22), we simplify the solution form in (19) by using the property

$$g_p'(\beta_i)=\epsilon+(\bar{Q}\boldsymbol{\beta}-\boldsymbol{y})_i,\ \text{and}\ g_n'(\beta_i)=-\epsilon+(\bar{Q}\boldsymbol{\beta}-\boldsymbol{y})_i. \tag{23}$$

Following the same technique in Section 3.2.1, we maintain a vector $\boldsymbol{u}$ and calculate $(\bar{Q}\boldsymbol{\beta})$ by

$$(\bar{Q}\boldsymbol{\beta})_i=\boldsymbol{u}^T\boldsymbol{x}_i+\lambda\beta_i,\ \text{where}\ \boldsymbol{u}=\sum_{i=1}^{l}\beta_i\boldsymbol{x}_i.$$

The new DCD method to solve (15) is sketched in Algorithm 3.

For the convergence, we show in Appendix A that Algorithm 3 is a special case of the general framework in Tseng and Yun (2009) for non-smooth separable minimization. Their Theorem 2(b) implies that Algorithm 3 converges in an at least linear rate.

Theorem 1 *For L1-loss and L2-loss SVR, if $\boldsymbol{\beta}^k$ is the k-th iterate generated by Algorithm 3, then $\{\boldsymbol{\beta}^k\}$ globally converges to an optimal solution $\boldsymbol{\beta}^*$. The convergence rate is at least linear: there are $0<\mu<1$ and an iteration k_0 such that*

$$f_B(\boldsymbol{\beta}^{k+1})-f_B(\boldsymbol{\beta}^*)\leq\mu(f_B(\boldsymbol{\beta}^k)-f_B(\boldsymbol{\beta}^*)),\forall k\geq k_0.$$

For the stopping condition and the shrinking procedure, we will mainly follow the setting in LIBLINEAR for L1-regularized classification. To begin, we study how to measure the violation of the optimality condition of (16) during the optimization procedure. From Figure 2(c), we see that

$$\text{if } 0 < \beta_i^* < U \text{ is optimal for (16), then } g_p'(\beta_i^*) = 0.$$

Thus, if $0 < \beta_i < U$, $|g_p'(\beta_i)|$ can be considered as the violation of the optimality. From Figure 2(b), we have that

$$\text{if } \beta_i^* = 0 \text{ is optimal for (16), then } g_n'(\beta_i^*) \le 0 \le g_p'(\beta_i^*).$$

Thus,

$$\begin{cases} g_n'(\beta_i) & \text{if } \beta_i = 0 \text{ and } g_n'(\beta_i) > 0, \\ -g_p'(\beta_i) & \text{if } \beta_i = 0 \text{ and } g_p'(\beta_i) < 0 \end{cases}$$

gives the violation of the optimality. After considering all situations, we know that

$$\beta_i \text{ is optimal for (16) if and only if } v_i = 0,$$

where

$$v_i \equiv \begin{cases} |g_n'(\beta_i)| & \text{if } \beta_i \in (-U, 0), \text{ or } \beta_i = -U \text{ and } g_n'(\beta_i) \le 0, \\ |g_p'(\beta_i)| & \text{if } \beta_i \in (0, U), \text{ or } \beta_i = U \text{ and } g_p'(\beta_i) \ge 0, \\ g_n'(\beta_i) & \text{if } \beta_i = 0 \text{ and } g_n'(\beta_i) \ge 0, \\ -g_p'(\beta_i) & \text{if } \beta_i = 0 \text{ and } g_p'(\beta_i) \le 0, \\ 0 & \text{otherwise.} \end{cases} \tag{24}$$

If β is unconstrained (i.e., $U = \infty$), then (24) reduces to the minimum-norm subgradient used in L1-regularized problems. Based on it, Yuan et al. (2010) derive their stopping condition and shrinking scheme. We follow them to use a similar stopping condition.

$$\|\boldsymbol{v}^k\|_1 < \epsilon_s \|\boldsymbol{v}^0\|_1, \tag{25}$$

where $\boldsymbol{v}^0$ and $\boldsymbol{v}^k$ are the initial violation and the violation in the k-th iteration, respectively. Note that $\boldsymbol{v}^k$'s components are sequentially obtained via (24) in l coordinate descent steps of the k-th iteration.

For shrinking, we remove bounded variables (i.e., $\beta_i = 0$, U, or $-U$) if they may not be changed at the final iterations. Following Yuan et al. (2010), we use a "tighter" form of the optimality condition to conjecture that a variable may have stuck at a bound. We shrink β_i if it satisfies one of the following conditions.

$$\beta_i = 0 \text{ and } g_n'(\beta_i) < -M < 0 < M < g_p'(\beta_i), \tag{26}$$

$$\beta_i = U \text{ and } g_p'(\beta_i) < -M, \text{ or} \tag{27}$$

$$\beta_i = -U \text{ and } g_n'(\beta_i) > M, \tag{28}$$

where

$$M \equiv \max_i v_i^{k-1} \tag{29}$$

is the maximal violation of the previous iteration. The condition (26) is equivalent to

$$\beta_i = 0 \text{ and } -\epsilon + M < (\bar{Q}\boldsymbol{\beta})_i - y_i < \epsilon - M. \tag{30}$$

Algorithm 4 Details of Algorithm 3 with a stopping condition and a shrinking implementation.

1. Given $\boldsymbol{\beta}$ and corresponding $\boldsymbol{u} = \sum_{i=1}^{l} \beta_i \boldsymbol{x}_i$.
2. Set $\lambda = 0$ and $U = C$ if L1-loss SVR; $\lambda = 1/(2C)$ and $U = \infty$ if L2-loss SVR.
3. Compute the Hessian diagonal $\bar{Q}_{ii},\ \forall i = 1, \ldots, l$.
4. $M \leftarrow \infty$, and compute $\|\boldsymbol{v}^0\|_1$ by (24).
5. $T \leftarrow \{1, \ldots, l\}$.
6. For $k = 0, 1, 2, \ldots$
   - 6.1. Randomly permute T.
   - 6.2. For $i \in T$ // select an index to update
     - 6.2.1. $g'_p \leftarrow -y_i + \boldsymbol{u}^T \boldsymbol{x}_i + \lambda\beta_i + \epsilon$, $g'_n \leftarrow -y_i + \boldsymbol{u}^T \boldsymbol{x}_i + \lambda\beta_i - \epsilon$.
     - 6.2.2. Find v_i^k by (24).
     - 6.2.3. If any condition in (26)–(28) is satisfied
       $T \leftarrow T \backslash \{i\}$.
       continue
     - 6.2.4. Find s by (22).
     - 6.2.5. $\boldsymbol{u} \leftarrow \boldsymbol{u} + (s - \beta_i)\boldsymbol{x}_i$.
     - 6.2.6. $\beta_i \leftarrow s$.
   - 6.3. If $\|\boldsymbol{v}^k\|_1 / \|\boldsymbol{v}^0\|_1 < \epsilon_s$
     If $T = \{1, \ldots, l\}$
       break
     else
       $T \leftarrow \{1, \ldots, l\}$, and $M \leftarrow \infty$.
   else
     $M \leftarrow \|\boldsymbol{v}^k\|_\infty$.

This is almost the same as the one used in Yuan et al. (2010); see Equation (32) in that paper. However, there are some differences. First, because they solve L1-regularized SVC, ϵ in (30) becomes the constant one. Second, they scale M to a smaller value. Note that M used in (26)–(28) controls how aggressive our shrinking scheme is. In Section 4.6, we will investigate the effect of using different M values.

For L2-loss SVR, α_i is not upper-bounded in the dual problem, so (26) becomes the only condition to shrink variables. This makes L2-loss SVR have less opportunity to shrink variables than L1-loss SVR. The same situation has been known for L2-loss SVC.

In Section 3.2.1, we pointed out some redundant operations in calculating the projected gradient of $f_A(\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-)$. If $0 < \beta_i < U$, we have $\alpha_i^+ = \beta_i$ and $\alpha_i^- = 0$. In this situation, Equation (13) indicates that for finding the maximal violation of the optimality condition, we only need to check $\nabla_i^P f_A(\boldsymbol{\alpha})$ rather than $\nabla_{i+l}^P f_A(\boldsymbol{\alpha})$. From (11) and (23),

$$\nabla_i^P f_A(\boldsymbol{\alpha}) = (\bar{Q}\boldsymbol{\beta} - \boldsymbol{y})_i + \epsilon = g'_p(\beta).$$

This is what we checked in (24) when $0 < \beta < U$. Therefore, no operations are wasted.

Algorithm 4 is the overall procedure to solve (15). In the beginning, we set $M = \infty$, so no variables are shrunk at the first iteration. The set T in Algorithm 4 includes variables which have not been shrunk. During the iterations, the stopping condition of a smaller problem of T is checked. If it is satisfied but T is not the full set of variables, we reset T to be $\{1, \ldots, l\}$; see the if-else statement in step 6.3 of Algorithm 4. This

setting ensures that the algorithm stops only after the stopping condition for problem (15) is satisfied. Similar approaches have been used in LIBSVM (Chang and Lin, 2011) and some solvers in LIBLINEAR.

3.3 Difference Between Linear and Nonlinear SVR

The discussion in Sections 3.2.1–3.2.2 concludes that α_i^+ and α_i^- should be solved together rather than separately. Interestingly, for nonlinear (kernel) SVR, Liao et al. (2002) argue that the opposite is better. They consider SVR with a bias term, so the dual problem contains an additional linear constraint.

$$\sum_{i=1}^{l}(\alpha_i^+ - \alpha_i^-) = 0.$$

Because of this constraint, their coordinate descent implementation (called decomposition methods in the SVM community) must select at least two variables at a time. They discuss the following two settings.

1. Considering $f_A(\boldsymbol{\alpha})$ and selecting $i, j \in \{1, \ldots, 2l\}$ at a time.

2. Selecting $i, j \in \{1, \ldots, l\}$ and then updating α_i^+, α_i^-, α_j^+, and α_j^- together. That is, a four-variable subproblem is solved.

The first setting corresponds to ours in Section 3.2.1, while the second is related to that in Section 3.2.2. We think Liao et al. (2002) prefer the first because of the following reasons, from which we can see some interesting differences between linear and nonlinear SVM.

1. For nonlinear SVM, we can afford to use gradient information for selecting the working variables; see reasons explained in Section 4.1 of Hsieh et al. (2008). This is in contrast to the sequential selection for linear SVM. Following the gradient-based variable selection, Liao et al. (2002, Theorem 3.4) show that if an optimal $(\alpha_i^+)^* > 0$, then α_i^- remains zero in the final iterations without being selected for update. The situation for $(\alpha_i^-)^* > 0$ is similar. Therefore, their coordinate descent algorithm implicitly has a shrinking implementation, so the first concern discussed in Section 3.2.1 is alleviated.

2. Solving a four-variable subproblem is complicated. In contrast, for the two-variable subproblem of α_i^+ and α_i^-, we demonstrate in Section 3.2.2 that a simple closed-form solution is available.

3. The implementation of coordinate descent methods for nonlinear SVM is more complicated than that for linear because of steps such as gradient-based variable selection and kernel-cache maintenance, etc. Thus, the first setting of minimizing $f_A(\boldsymbol{\alpha})$ possesses the advantage of being able to reuse the code of SVC. This is the approach taken by the nonlinear SVM package LIBSVM (Chang and Lin, 2011), in which SVC and SVR share the same optimization solver. In contrast, for linear SVC/SVR, the implementation is simple, so we can have a dedicated code for SVR. In this situation, minimizing $f_B(\boldsymbol{\beta})$ is more preferable than $f_A(\boldsymbol{\alpha})$.

4 Experiments

In this section, we compare nonlinear/linear SVR and evaluate the methods described in Sections 3. Two evaluation criteria are used. The first one is mean squared error (MSE).

$$\text{mean squared error } = \sum_{i=1}^{l} (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2.$$

The other is squared correlation coefficient (R^2). Given the target values $\boldsymbol{y}$ and the predicted values $\boldsymbol{y}'$, R^2 is defined as

$$\frac{\left(\sum_i (y'_i - E[y'_i])(y_i - E[y_i])\right)^2}{\sigma^2_{\boldsymbol{y}} \sigma^2_{\boldsymbol{y}'}} = \frac{\left(l \sum_i y'_i y_i - (\sum_i y'_i)(\sum_i y_i)\right)^2}{\left(l \sum_i y_i^2 - (\sum_i y_i)^2\right)\left(l \sum_i y_i'^2 - (\sum_i y'_i)^2\right)}.$$

4.1 Experimental Settings

We consider the following data sets in our experiments. All except CTR are publicly available at LIBSVM data set.[2]

- MSD: We consider this data because it is the largest regression set in the UCI Machine Learning Repository (Frank and Asuncion, 2010). It is originally from Bertin-Mahieux et al. (2011). Each instance contains the audio features of a song, and the target value is the year the song was released. The original target value is from 1922 to 2011, but we follow Bertin-Mahieux et al. (2011) to linearly scale it to $[0, 1]$.
- TFIDF-2006, LOG1P-2006: This data set comes from some context-based analysis and discussion of the financial condition of a corporation (Kogan et al., 2009).[3] The target values are the log transformed volatilities of the corporation. We use records in the last year (2006) as the testing data, while the previous five years (2001–2005) for training.
  There are two different feature representations. TFIDF-2006 contains TF-IDF (term frequency and inverse document frequency) of unigrams, but LOG1P-2006 contains
  $$\log(1 + \text{TF}),$$
  where TF is the term frequency of unigrams and bigrams. Both representations also include the volatility in the past 12 months as an additional feature.
- CTR: The data set is from an Internet company. Each feature vector is a binary representation of a web page and an advertisement block. The target value is the click-through-rate (CTR) defined as (#clicks)/(#page views).
- KDD2010b: This classification problem comes from KDD Cup 2010. The class label indicates whether a student answered a problem correctly or not on a online tutoring system. We consider this problem because of several reasons. First, we have not found other large and sparse regression problems. Second, we are interested in the performance of SVR algorithms when a classification problem is treated as a regression one.

[2] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

[3] The raw data are available at `http://www.ark.cs.cmu.edu/10K/`.

Table 1: Data set statistics: #non-zeros means the number of non-zero elements in all training instances. Note that data sets are sorted according to the number of features.

Data	#instances		#features	#non-zeros	range of $\boldsymbol{y}$
	training	testing		in training	
MSD	463,715	51,630	90	41,734,346	$[0, 1]$
TFIDF-2006	16,087	3,308	150,360	19,971,015	$[-7.90, -0.52]$
LOG1P-2006	16,087	3,308	4,272,227	96,731,839	$[-7.90, -0.52]$
CTR	11,382,195	208,988	22,510,600	257,526,282	$[0, 1]$
KDD2010b	19,264,097	748,401	29,890,095	566,345,888	$\{0, 1\}$

The numbers of instances, features, nonzero elements in training data, and the range of target values are listed in Table 1. Except MSD, all others are large sparse data.

We use the zero vector as the initial solution of all algorithms. All implementations are in C++ and experiments are conducted on a 64-bit machine with Intel Xeon 2.0GHz CPU (E5504), 4MB cache, and 32GB main memory. Programs used for our experiment can be found at `http://www.csie.ntu.edu.tw/~cjlin/liblinear/exp.html`.

4.2 A Comparison Between Two DCD Algorithms

Our first experiment is to compare two DCD implementations (Algorithms 2 and 4) so that only the better one is used for subsequence analysis. For this comparison, we normalize each instance to a unit vector and consider L1-loss SVR.

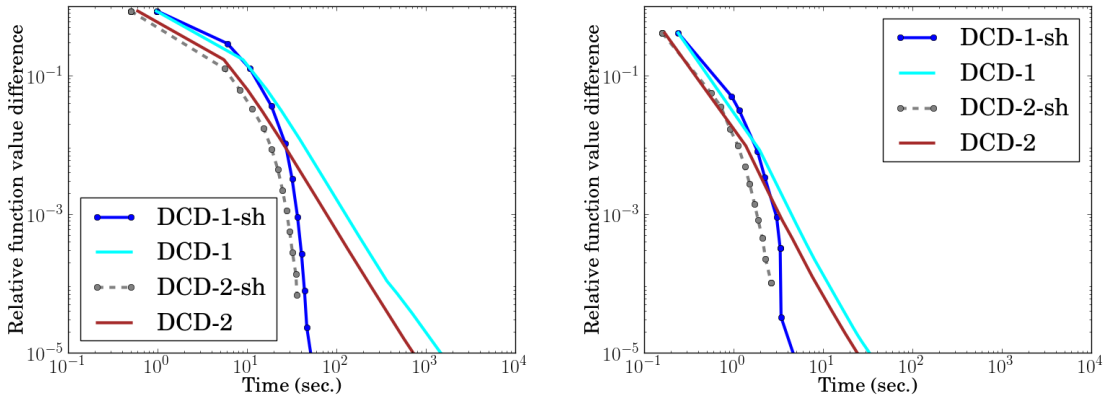
Figure 3 presents results using parameters $C = 1$ and $\epsilon = 0.1$. The x-axis is the training time, and the y-axis is the relative difference to the dual optimal function value.

$$\frac{f_A(\boldsymbol{\alpha}) - f_A(\boldsymbol{\alpha}^*)}{|f_A(\boldsymbol{\alpha}^*)|}, \tag{31}$$

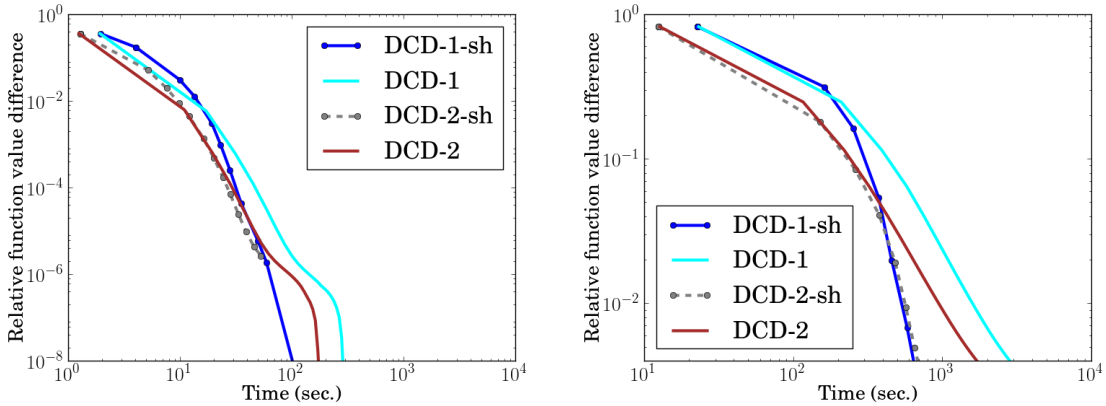
where $\boldsymbol{\alpha}^*$ is the optimum solution. We run optimization algorithms long enough to get an approximate $f_A(\boldsymbol{\alpha}^*)$. In Figure 3, DCD-1 and DCD-1-sh are Algorithm 2 without/with shrinking, respectively. DCD-2, and DCD-2-sh are the proposed Algorithm 4. If shrinking is not applied, we simply plot the value (31) once every eight iterations. With shrinking, the setting is more complicated because the stopping tolerance ϵ_s affects the shrinking implementation; see step 6.3 in Algorithm 4. Therefore, we run Algorithms 2 and 4 several times under various ϵ_s values to obtain pairs of (training time, function value).

Results show that DCD-2 is faster than DCD-1. Because the training time (x-axis) is in log-scale, the difference is significant. This observation is consistent with our discussion in Section 3.2.1 that Algorithm 2 suffers from some redundant operations. We mentioned that shrinking can reduce the overhead and this is supported by the result that DCD-1-sh becomes closer to DCD-2-sh. Based on this experiment, we only use Algorithm 4 in subsequent analysis.

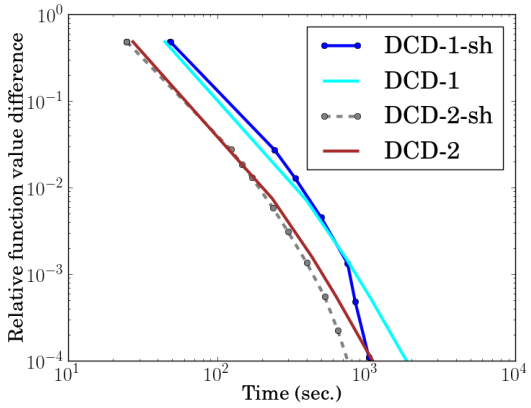
This experiment also reveals how useful the shrinking technique is. For both Algorithms 2 and 4, we clearly observe that shrinking very effectively reduces the training time.



(a) MSD (b) TFIDF-2006



(c) LOG1P-2006 (d) CTR



(e) KDD2010b

Figure 3: Relative difference to the dual optimal function values of L1-loss SVR using $C = 1$ and $\epsilon = 0.1$. Data instances are normalized to unit vectors. DCD-1-sh and DCD-2-sh are DCD-1 and DCD-2 with shrinking, respectively. Both x-axis and y-axis are in log scale.

Table 2: Testing MSE and training time using DCD for linear SVR and LIBSVM for nonlinear SVR with RBF kernel. Parameter selection is conducted by five-fold CV. Time means training time in seconds. Because LIBSVM's running time is long, for some data, we use only subsets for training.

Data (percentage for training)	Linear (DCD)				RBF (LIBSVM)				
	ϵ	C	MSE	time	ϵ	C	γ	MSE	time
MSD (1%)	2^{-4}	2^{5}	0.0153	2.38	2^{-4}	2^{5}	2^{-3}	0.0129	4.86
TFIDF-2006	2^{-10}	2^{6}	0.2031	26.29	2^{-6}	2^{6}	2^{0}	0.1965	3,847.42
LOG1P-2006	2^{-4}	2^{1}	0.1422	12.14	2^{-10}	2^{1}	2^{0}	0.1381	16,046.7
CTR (0.1%)	2^{-6}	2^{-3}	0.0296	0.05	2^{-8}	2^{-2}	2^{0}	0.0294	15.19
KDD2010b (0.1%)	2^{-4}	2^{-1}	0.0979	0.07	2^{-6}	2^{0}	2^{0}	0.0941	95.07

4.3 A Comparison Between Linear and Nonlinear SVR

We wrote in Section 1 that the motivation of this research work is to check if for some applications linear SVR can give competitive MSE/R^2 with nonlinear SVR, but enjoy faster training. In this section, we compare DCD for linear SVR with the package LIBSVM (Chang and Lin, 2011) for nonlinear SVR.

For LIBSVM, we consider RBF kernel, so Q_{ij} in Equation (5) becomes

$$Q_{ij} \equiv e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2},$$

where γ is a user-specified parameter. Because LIBSVM's training time is very long, we only use 1% training data for MSD, and 0.1% training data for CTR and KDD2010b. We conduct five-fold cross validation to find the best $C \in \{2^{-4}, 2^{-3}, \ldots, 2^{6}\}$, $\epsilon \in \{2^{-10}, 2^{-8}, \ldots, 2^{-2}\}$, and $\gamma \in \{2^{-8}, 2^{-7}, \ldots, 2^{0}\}$. For LIBSVM, we assign 16GB memory space for storing recently used kernel elements (called kernel cache). We use stopping tolerance 0.1 for both methods although their stopping conditions are slightly different. Each instance is normalized to a unit vector.

In Table 2, we observe that for all data sets except MSD, nonlinear SVR gives only marginally better MSE than linear SVR, but the training time is prohibitively long. Therefore, for these data sets, linear SVR is more appealing than nonlinear SVR.

4.4 A Comparison Between TRON and DCD on Data with/without Normalization

TRON and DCD are the two methods discussed in Section 3 for training linear SVR. We compare them in this section. We also check if their behavior is similar to when they are applied to linear SVC. Because TRON is not applicable to L1-loss SVR, L2-loss SVR is considered here.

A common practice in document classification is to normalize each feature vector to have unit length. Because the resulting optimization problem may have a better numerical condition, this normalization procedure often helps to shorten the training time. We will investigate its effectiveness for regression data.

We begin with comparing TRON and DCD on the original data without normalization. Figure 4 shows training time versus the relative difference to the optimal primal

Table 3: MSE without and with data normalization. Parameter selection is only applied to the normalized data because the running time is too long for the original data.

Data	Original $C = 1$, $\epsilon = 0.1$ MSE	Normalized $C = 1$, $\epsilon = 0.1$ MSE	Normalized + parameter selection C	ϵ	MSE
MSD	0.01474	0.01651	2^6	2^{-10}	0.01573
TFIDF-2006	0.15170	0.38568	2^6	2^{-10}	0.38315
LOG1P-2006	0.15606	0.15192	2^2	2^{-10}	0.13882
CTR	0.02951	0.03020	2^4	2^{-8}	0.02852
KDD2010b	0.08350	0.08206	2^{-1}	2^{-10}	0.07750

function value.

$$\frac{f(\boldsymbol{w}) - f(\boldsymbol{w}^*)}{|f(\boldsymbol{w}^*)|}.$$

Although DCD solves the dual problem, we can obtain a corresponding primal value using $\boldsymbol{w} = X^T\boldsymbol{\beta}$. Primal values obtained in this way may not be decreasing, so DCD's curves in Figure 4 fluctuate. Because practically users apply TRON or DCD under a fixed stopping tolerance, we draw two horizontal lines in Figure 4 to indicate the result using a typical tolerance value. We use $\epsilon_s = 0.001$ in (8) and $\epsilon_s = 0.1$ in (25).

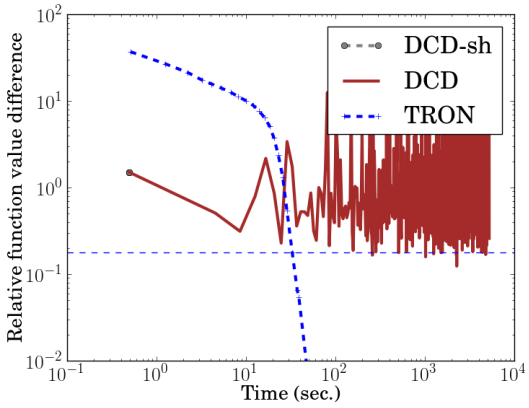
We observe that DCD outperforms TRON when data have more features. For MSD, which has only 90 features, DCD's primal function value is so unstable that it does not reach the stopping condition for drawing the horizontal line. A primal method like TRON is more suitable for this data set because of the smaller number of variables. In contrast, KDD2010b has 29 million features, and DCD is much more efficient than TRON. This result is consistent with the situation in classification (Hsieh et al., 2008).

Figures 5 and 6 present testing MSE and R^2, respectively. Similar to the results in Figure 4, DCD is better than TRON for data with more features.

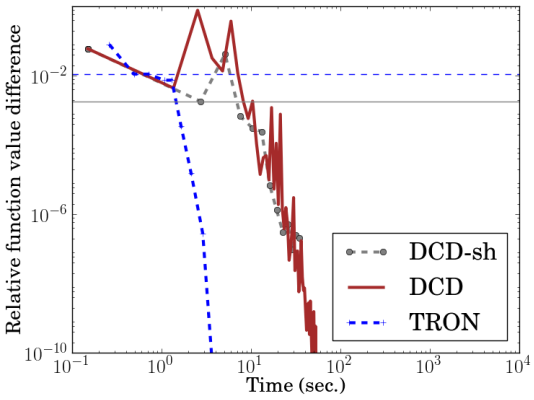
Next, we compare TRON and DCD on data normalized to have unit length. Results of function values, testing MSE, and testing R^2 are respectively shown in Figures 7–9. From Figures 4 and 7, both methods have shorter training time for normalized data. For example, for CTR, DCD is 10 times faster, while TRON is 1.6 times faster. DCD becomes very fast for all problems including MSD. Therefore, like the classification case, if data have been properly normalized, DCD is generally faster than TRON.

We can compare Figures 3 and 7 to check the effect of shrinking for L1-loss and L2-loss SVR. Clearly, shrinking is more useful for L1-loss SVR as discussed in Section 3.2.2.

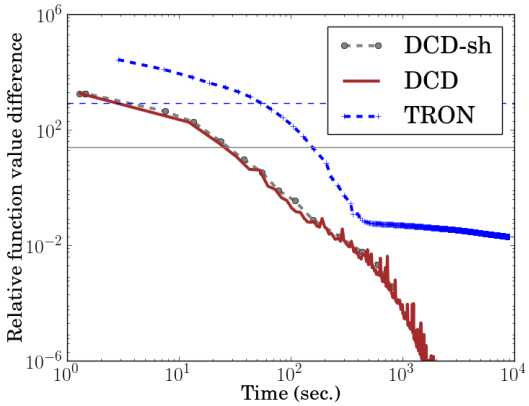
To compare the testing performance without/with data normalization, we show MSE in Table 3. We use TRON because DCD fails on MSD if it is not normalized. An issue of the comparison between Figures 4 and 7 is that we use $C = 1$ and $\epsilon = 0.1$ without parameter selection. We tried to conduct parameter selection but can only report results of the normalized data. The running time is too long for the original data. From Table 3, except TFIDF-2006, normalization does not cause inferior MSE values. This result indicates that for the practical use of linear SVR, data normalization is a useful preprocessing procedure.



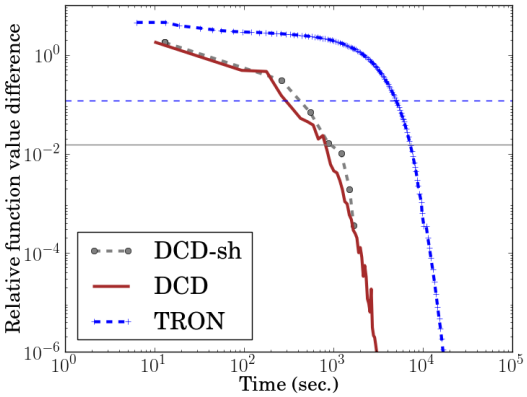
(a) MSD



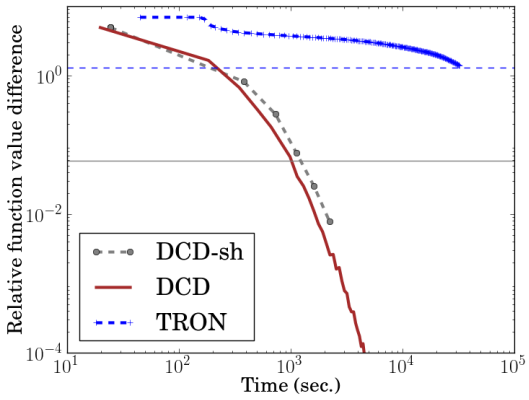
(b) TFIDF-2006



(c) LOG1P-2006

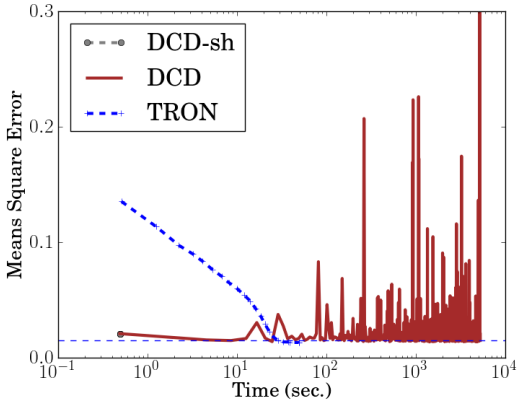


(d) CTR

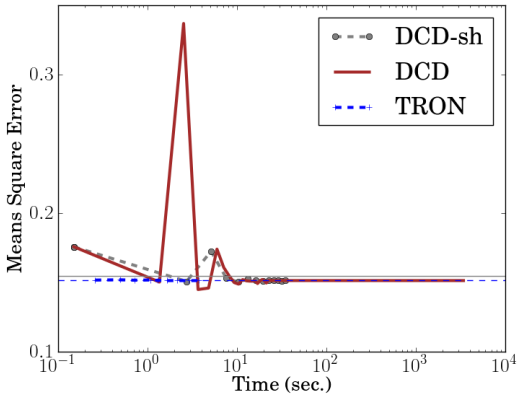


(e) KDD2010b

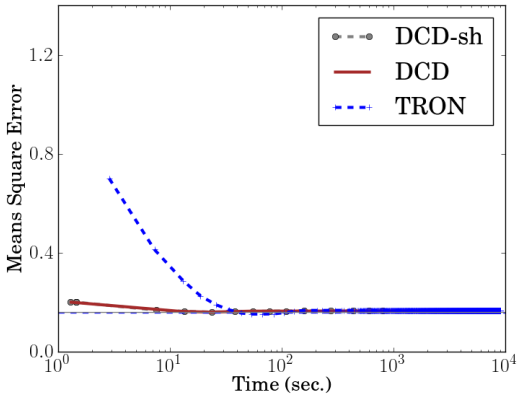
Figure 4: Relative differences to the optimal primal function value of L2-loss SVR using $C = 1$ and $\epsilon = 0.1$. Original data without normalization are used. The dotted and solid horizontal lines respectively indicate the function values of TRON using $\epsilon_s = 0.001$ in (8) and DCD using $\epsilon_s = 0.1$ in (25).



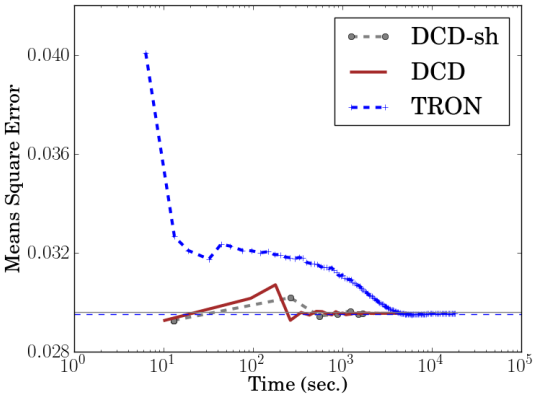
(a) MSD



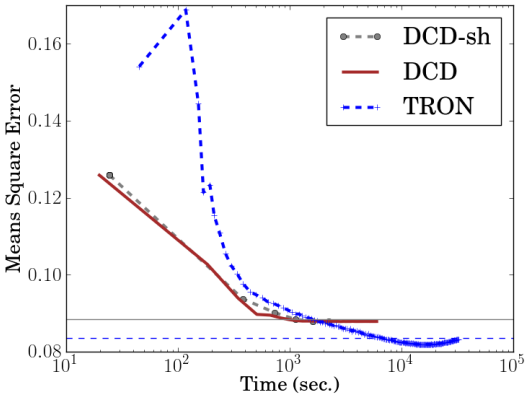
(b) TFIDF-2006



(c) LOG1P-2006

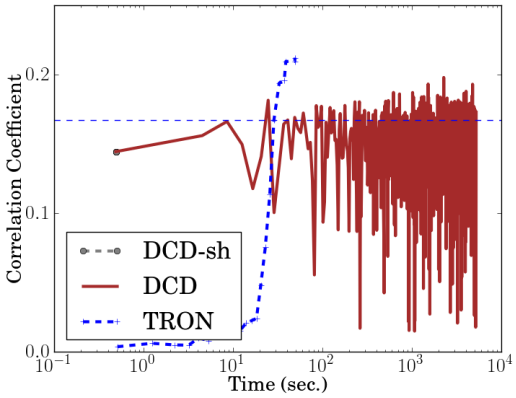


(d) CTR

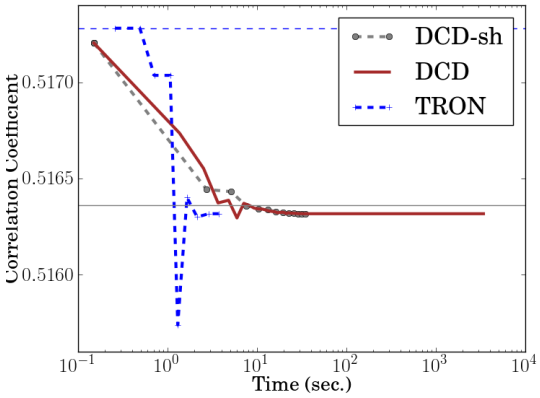


(e) KDD2010b

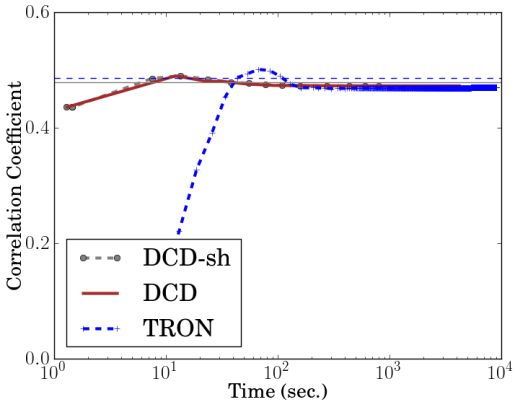
Figure 5: MSE of L2-loss SVR. Settings are the same as those in Figure 4.



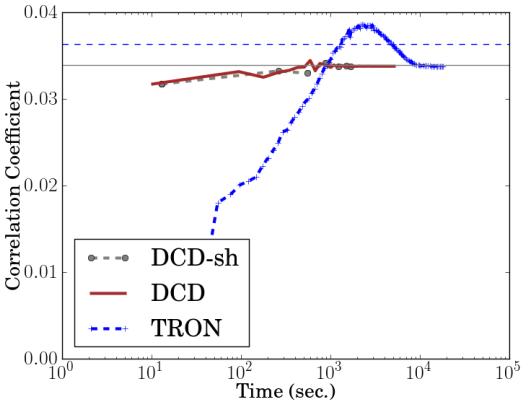
(a) MSD



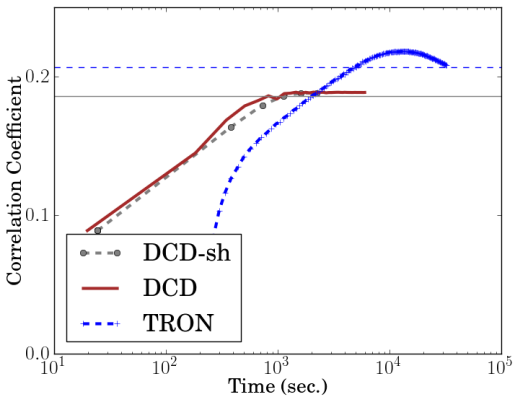
(b) TFIDF-2006



(c) LOG1P-2006

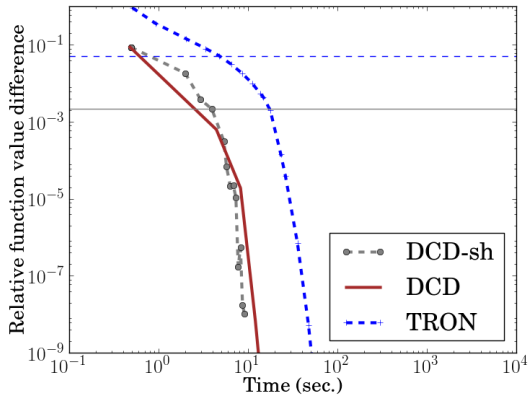


(d) CTR

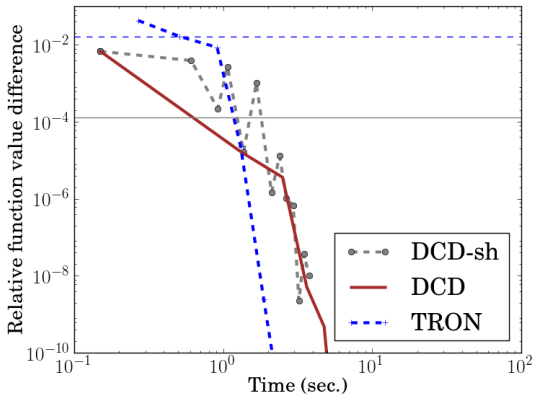


(e) KDD2010b

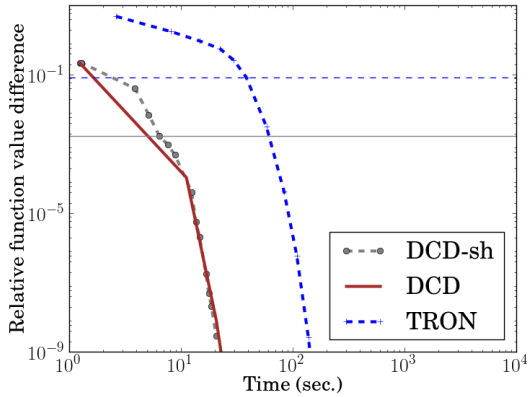
Figure 6: R^2 of L2-loss SVR. Settings are the same as those in Figure 4.



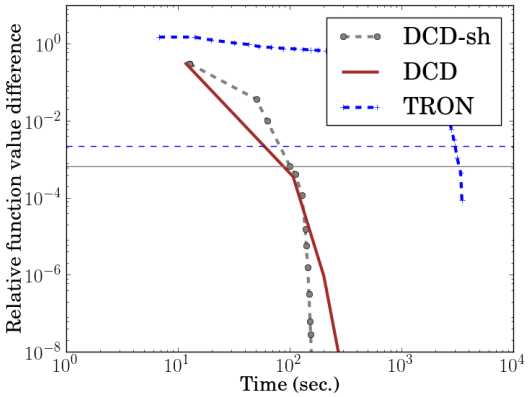
(a) MSD



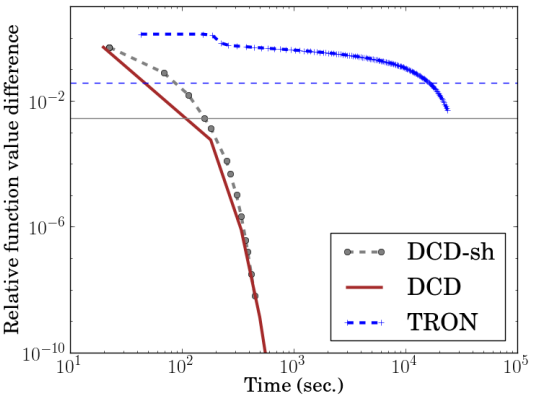
(b) TFIDF-2006



(c) LOG1P-2006

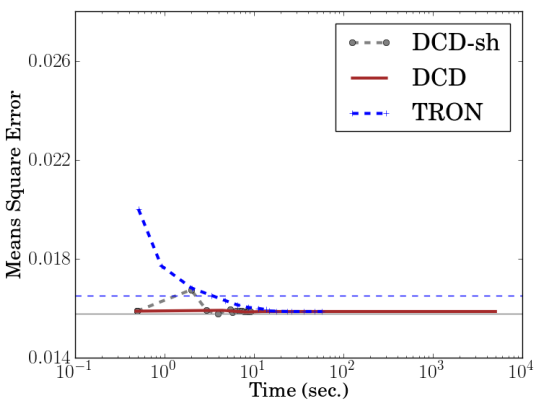


(d) CTR

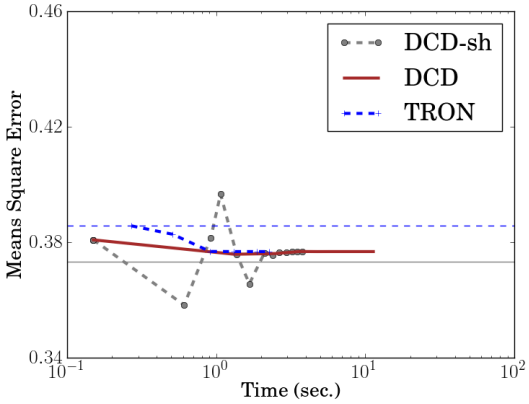


(e) KDD2010b

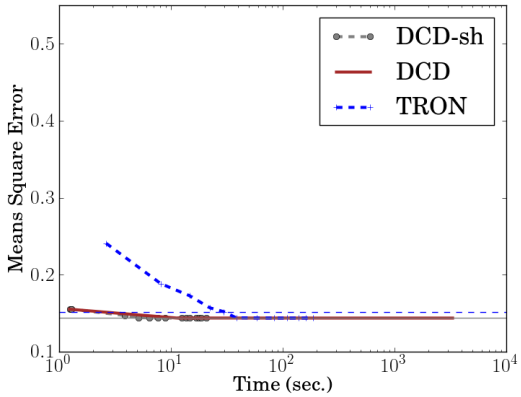
Figure 7: Relative differences to the optimal primal function value of L2-loss SVR using $C = 1$ and $\epsilon = 0.1$. Data instances are normalized to unit vectors. The dotted and solid horizontal lines respectively indicate the function values of TRON using $\epsilon_s = 0.001$ in (8) and DCD using $\epsilon_s = 0.1$ in (25).



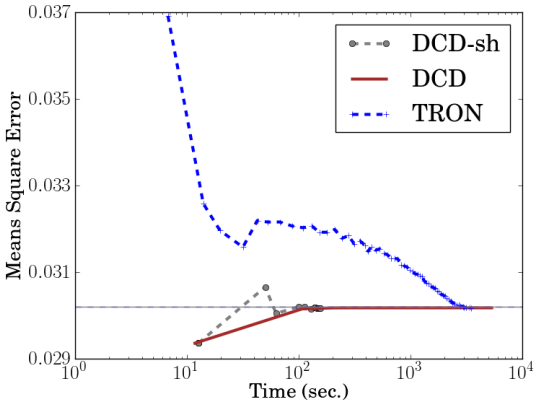
(a) MSD



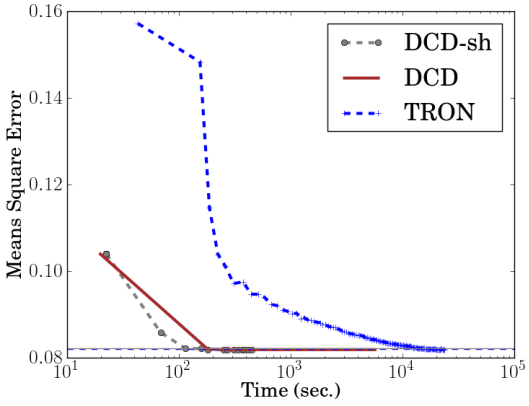
(b) TFIDF-2006



(c) LOG1P-2006

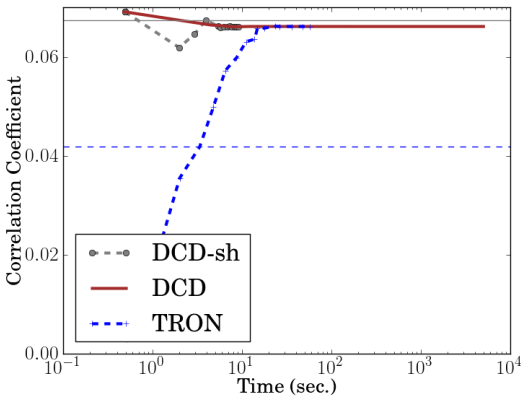


(d) CTR

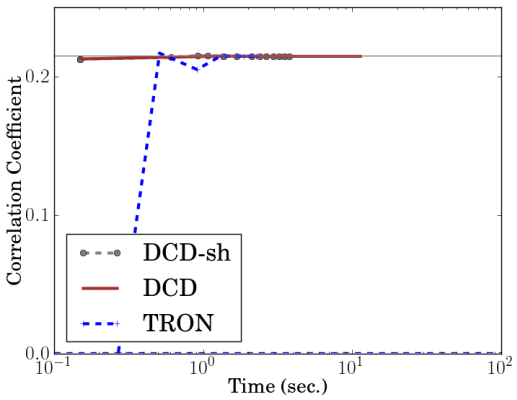


(e) KDD2010b

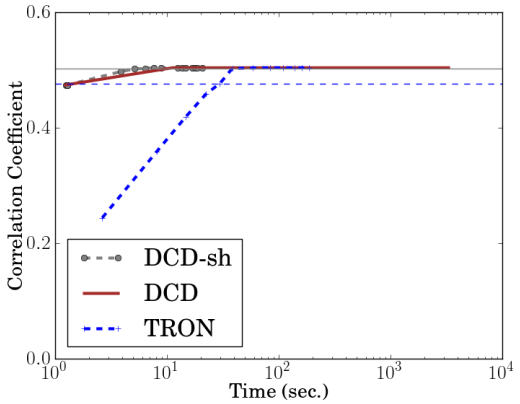
Figure 8: MSE of L2-loss SVR. Settings are the same as those in Figure 7.



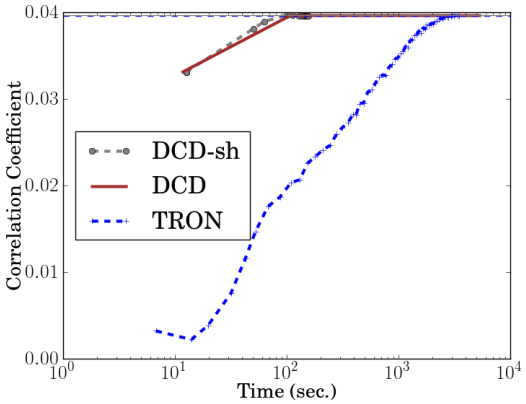
(a) MSD



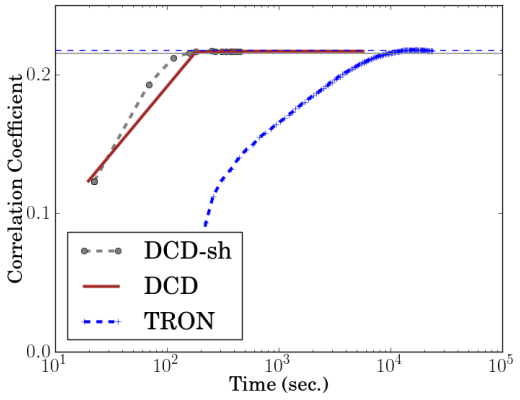
(b) TFIDF-2006



(c) LOG1P-2006



(d) CTR



(e) KDD2010b

Figure 9: R^2 of L2-loss SVR. Settings are the same as those in Figure 7.

Table 4: MSE of L2-loss SVR with and without the bias term.

Data	without bias	with bias
MSD	0.01651	0.01479
TFIDF-2006	0.38568	0.38573
LOG1P-2006	0.15192	0.16919
CTR	0.03020	0.03031
KDD2010b	0.08206	0.09699

4.5 With and Without the Bias Term in the SVR Prediction Function

We omit the bias term in the above discussion because we suspect that it has little effect on the performance. LIBLINEAR supports a common way to include a bias term by appending one more feature to each data instance.

$$\boldsymbol{x}_i^T \leftarrow [\boldsymbol{x}_i^T, 1] \quad \boldsymbol{w}^T \leftarrow [\boldsymbol{w}^T, b].$$

We apply TRON on normalized data sets to compare MSE values with and without the bias term. With the stopping tolerance $\epsilon_s = 0.001$, the results in Table 4 show that MSE values obtained with/without the bias term are not very different. Results in Table 2 also support this finding because LIBSVM solves SVR with a bias term.

4.6 Aggressiveness of DCD's Shrinking Scheme

In Section 3.2.2, we introduced DCD's shrinking scheme with a parameter M defined as the maximal violation of the optimality condition. We pointed out that the smaller M is, the more aggressive the shrinking method is. To check if choosing M by the way in (29) is appropriate, we compare the following settings.

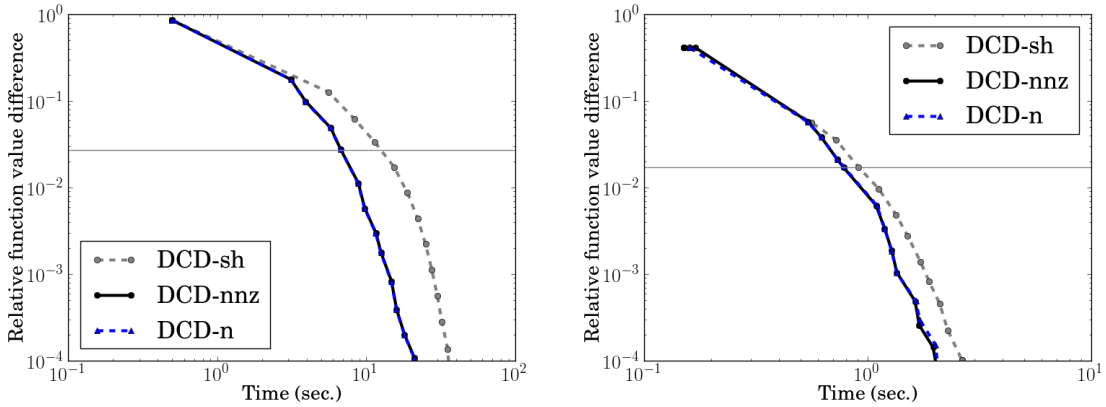
1. DCD-sh: The method in Section 3.2.2 using M defined in (29).
2. DCD-nnz: M is replaced by $M/\bar{n}$, where $\bar{n}$ is the average number of non-zero feature values per instance.
3. DCD-n: M is replaced by M/n.

Because

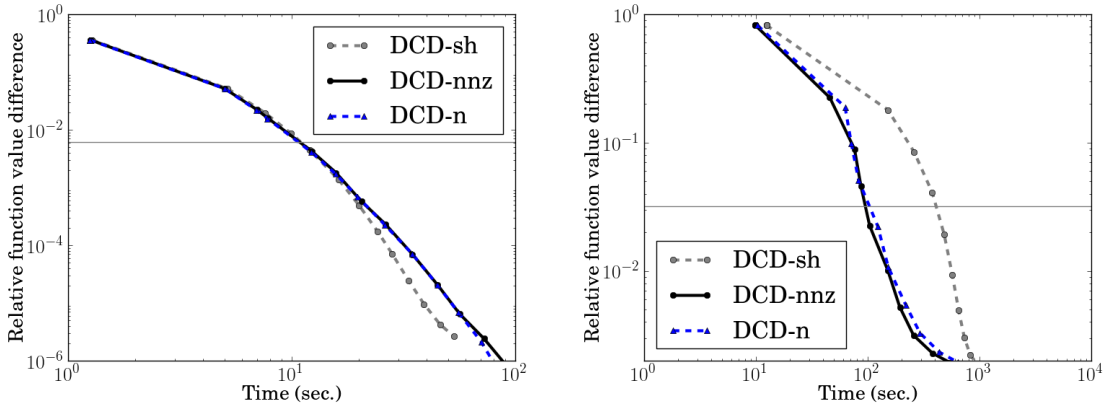
$$\frac{M}{n} < \frac{M}{\bar{n}} < M,$$

DCD-n is the most aggressive setting, while DCD-sh is the most conservative.

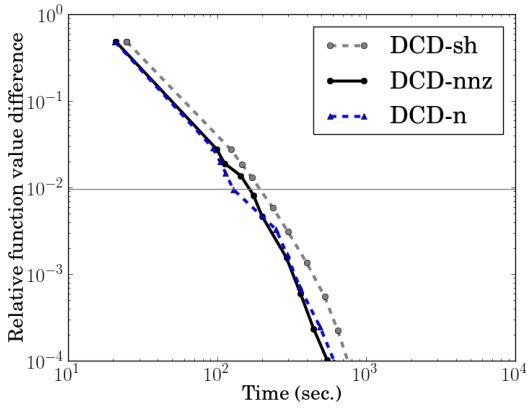
Using L1-loss SVR, Figure 10 shows the relationship between the relative difference to the optimal dual function value and the training time. Results indicate that most data sets need a more aggressive shrinking strategy. However, if L2-loss SVR is applied instead, Figure 11 shows different results. Except CTR, aggressive shrinking strategies make the results worse. A possible reason is that less variables are shrunk for L2-loss SVR (see explanation in Sections 3.2.2 and 4.4), so an aggressive strategy may wrongly shrink some variables.



(a) MSD (b) TFIDF-2006

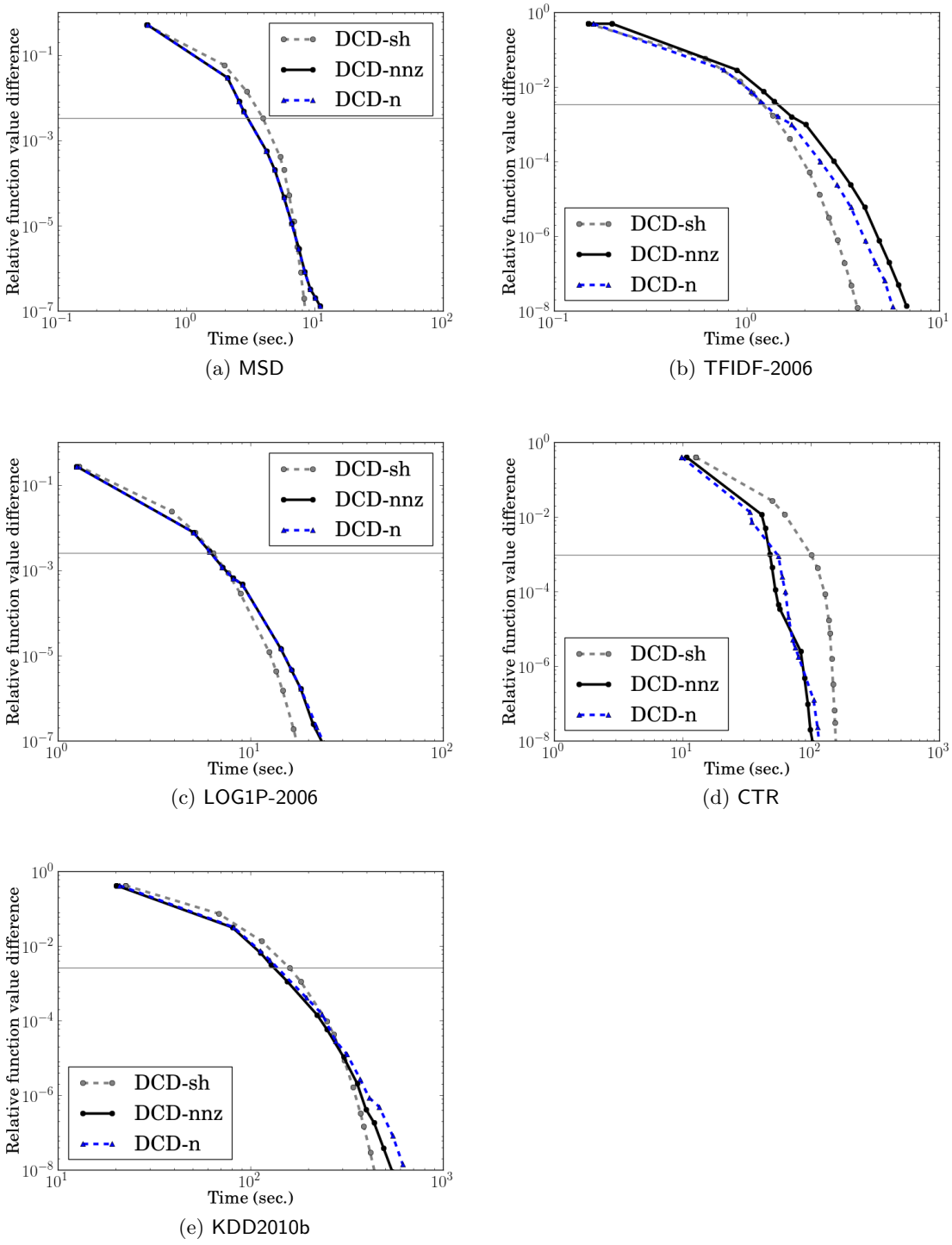


(c) LOG1P-2006 (d) CTR



(e) KDD2010b

Figure 10: A comparison of three shrinking settings. L1-SVR with $C = 1$ and $\epsilon = 0.1$ is applied on normalized data. We show the relative difference to the dual optimal function values and training time (in seconds).



(a) MSD (b) TFIDF-2006

(c) LOG1P-2006 (d) CTR

(e) KDD2010b

Figure 11: A comparison of three shrinking settings. L2-SVR is used. Other settings are the same as those in Figure 10.

Table 5: L1-/L2-loss SVR and regularized least-square regression. Regularized least-square regression is the same as L2-loss SVR with $\epsilon = 0$.

Data	L1-loss SVR				L2-loss SVR				Least square		
	ϵ	C	MSE	time	ϵ	C	MSE	time	C	MSE	time
MSD	2^{-4}	2^{1}	0.0149	24.1	2^{-10}	2^{6}	0.0149	160.9	2^{1}	0.0147	7.1
TFIDF-2006	2^{-10}	2^{6}	0.2030	33.2	2^{-10}	2^{6}	0.2251	36.7	2^{6}	0.2250	37.2
LOG1P-2006	2^{-4}	2^{1}	0.1421	17.0	2^{-10}	2^{2}	0.1389	14.1	2^{1}	0.1383	9.8
CTR	2^{-6}	2^{-1}	0.0287	306.7	2^{-8}	2^{4}	0.0287	1049.9	2^{0}	0.0283	119.8
KDD2010b	2^{-4}	2^{1}	0.0826	253.9	2^{-10}	2^{-1}	0.0775	105.7	2^{-1}	0.0775	105.2

5 Discussions and Conclusions

In this paper, we extend LIBLINEAR's SVC solvers TRON and DCD to solve large-scale linear SVR problems. The extension for TRON is straightforward, but is not trivial for DCD. We propose an efficient DCD method to solve a reformulation of the dual problem. Experiments show that many properties of TRON and DCD for SVC still hold for SVR.

An interesting future research direction is to apply coordinate descent methods for L1-regularized least-square regression, which has been shown to be related to problem (15). However, we expect some differences because the former is a primal problem, while the latter is a dual problem.

If $\epsilon = 0$, L2-loss SVR is reduced to regularized least-square regression (also called ridge regression by Hoerl and Kennard, 1970). The dual problem (15) becomes a simple unconstrained quadratic problem. Both our TRON and DCD implementations can be applied to the situation of $\epsilon = 0$, so we conduct a preliminary comparison between linear SVR and least-square regression in Table 5. Regularized least-square regression gives similar MSE to L2-loss SVR. This result seems to indicate that for these data sets, if L2 loss is considered, there is no need to consider the ϵ parameter. For L1-loss SVR and least square regression, the difference is more significant because of different loss functions. Whether similar observations hold for other large sparse data is an interesting research problem. Due to the lack of large regression data sets, we have not been able to conduct more experiments. We hope this work can motivate more studies and more public data in the near future.

In summary, we have successfully demonstrated that for some document data, the proposed methods can efficiently train linear SVR, while achieve comparable testing errors to nonlinear SVR. Based on this study, we have expanded the package LIBLINEAR (after version 1.9) to support large-scale linear SVR.

A Linear Convergence of Algorithm 3

To apply results in Tseng and Yun (2009), we first check if problem (15) is covered in their study. Tseng and Yun (2009) consider

$$\min_{\boldsymbol{\beta}} F(\boldsymbol{\beta}) + \epsilon P(\boldsymbol{\beta}), \tag{32}$$

where $F(\boldsymbol{\beta})$ is a smooth function and $P(\boldsymbol{\beta})$ is proper, convex, and lower semicontinuous. We can write (15) in the form of (32) by defining

$$F(\boldsymbol{\beta}) \equiv \frac{1}{2}\boldsymbol{\beta}^T\bar{Q}\boldsymbol{\beta} - \boldsymbol{y}^T\boldsymbol{\beta}, \quad \text{and} \quad P(\boldsymbol{\beta}) \equiv \begin{cases} \|\boldsymbol{\beta}\|_1 & \text{if } -U \le \beta_i \le U, \forall i, \\ \infty & \text{otherwise.} \end{cases}$$

Both $F(\boldsymbol{\beta})$ and $P(\boldsymbol{\beta})$ satisfy the required conditions. Tseng and Yun (2009) propose a general coordinate descent method. At each step certain rules are applied to select a subset of variables for update. Our rule of going through all l indices in one iteration is a special case of "Gauss-Seidel" rules discussed in their paper. If each time only one variable is updated, the subproblem of their coordinate descent method is

$$\min_s \begin{cases} \nabla_i F(\boldsymbol{\beta})(s-\beta_i) + \frac{1}{2}H(s-\beta_i)^2 + \epsilon|s| & \text{if } -U \le s \le U, \\ \infty & \text{otherwise,} \end{cases} \tag{33}$$

where H is any positive value. Because we use $H = \bar{Q}_{ii}$, if $\bar{Q}_{ii} > 0$, then (16) is a special case of (33). We will explain later that $\bar{Q}_{ii} = 0$ is not a concern.

Next, we check conditions and assumptions required by Theorem 2(b) of Tseng and Yun (2009). The first one is

$$\|\nabla F(\boldsymbol{\beta}_1) - \nabla F(\boldsymbol{\beta}_2)\| \le L\|\boldsymbol{\beta}_1 - \boldsymbol{\beta}_2\|, \forall \boldsymbol{\beta}_1, \boldsymbol{\beta}_2 \in \{\boldsymbol{\beta} \mid F(\boldsymbol{\beta}) < \infty\}.$$

Because $F(\boldsymbol{\beta})$ is a quadratic function, this condition easily holds by setting the largest eigenvalue of $\bar{Q}$ as L. We then check Assumption 1 of Tseng and Yun (2009), which requires that

$$\underline{\lambda} \le \bar{Q}_{ii} \le \bar{\lambda}, \forall i, \tag{34}$$

where $\underline{\lambda} > 0$. The only situation that (34) fails is when $\boldsymbol{x}_i = 0$ and L1-loss SVR is applied. In this situation, $\boldsymbol{\beta}^T\bar{Q}\boldsymbol{\beta}$ is not related to β_i and the minimization of $-y_i\beta_i + \epsilon|\beta_i|$ shows that the optimal β_i^* is

$$\beta_i^* = \begin{cases} U & \text{if } -y_i + \epsilon < 0, \\ -U & \text{if } -y_i - \epsilon > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{35}$$

We can remove these variables before applying DCD, so (34) is satisfied.[4]

For Assumption 2 in Tseng and Yun (2009), we need to show that the solution set of (32) is not empty. For L1-loss SVR, following Weierstrass' Theorem, the compact feasible domain ($\boldsymbol{\beta} \in [-U, U]^l$) implies the existence of optimal solutions. For L2-loss SVR, the strictly quadratic convex $F(\boldsymbol{\beta})$ implies that $\{\boldsymbol{\beta} \mid F(\boldsymbol{\beta}) + \epsilon P(\boldsymbol{\beta}) \le F(\boldsymbol{\beta}^0) + \epsilon P(\boldsymbol{\beta}^0)\}$ is compact, where $\boldsymbol{\beta}^0$ is any vector. Therefore, the solution set is also nonempty. Then Lemma 7 in Tseng and Yun (2009) implies that a quadratic $L(\boldsymbol{\beta})$ and a polyhedral $P(\boldsymbol{\beta})$ make their Assumption 2 hold.

[4] Actually, we do not remove these variables. Under the IEEE floating-point arithmetic, at the first iteration, the first two cases in (22) are ∞ and $-\infty$, respectively. Then, (21) projects the value back to U and $-U$, which are the optimal value shown in (35). Therefore, variables corresponding to $\boldsymbol{x}_i = \mathbf{0}$ have reached their optimal solution at the first iteration. Because they will never be updated, it is like that they have been removed.

Finally, by Theorem 2(b) of Tseng and Yun (2009), $\{\boldsymbol{\beta}^k\}$ generated by Algorithm 3 globally converges and $\{f_B(\boldsymbol{\beta}^k)\}$ converges at least linearly.

Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grant 98-2221-E-002-136-MY3.

References

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *In Proceedings of the Twelfth International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.

Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Corina Cortes and Vladimir Vapnik. Support-vector network. *Machine Learning*, 20: 273–297, 1995.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`.

Andrew Frank and Arthur Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf`.

Thorsten Joachims. Making large-scale SVM learning practical. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 169–184, Cambridge, MA, 1998. MIT Press.

Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

S. Sathiya Keerthi and Dennis DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6:341–361, 2005.

Shimon Kogan, Dimitry Levin, Bryan R. Routledge, Jacob S. Sagi, and Noah A. Smith. Predicting risk from financial reports with regression. In *In Proceedings of the North American Association for Computational Linguistics Human Language Technologies Conference*, pages 272–280, 2009.

Shuo-Peng Liao, Hsuan-Tien Lin, and Chih-Jen Lin. A note on the decomposition methods for support vector regression. *Neural Computation*, 14:1267–1281, 2002.

Chih-Jen Lin and Jorge J. Moré. Newton's method for large-scale bound constrained problems. *SIAM Journal on Optimization*, 9:1100–1127, 1999.

Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9:627–650, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf`.

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

John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: primal estimated sub-gradient solver for SVM. In *Proceedings of the Twenty Fourth International Conference on Machine Learning (ICML)*, 2007.

Paul Tseng and Sangwoon Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117:387–423, 2009.

Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, NY, 1995.

Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *Journal of Machine Learning Research*, 11:3183–3234, 2010. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/l1.pdf`.