Machine Learning (2023) 112:2975–3002
https://doi.org/10.1007/s10994-021-05996-7

Unified SVM algorithm based on LS-DC loss

Shuisheng Zhou[1] · **Wendi Zhou**[2]

Received: 2 September 2020 / Revised: 7 April 2021 / Accepted: 11 May 2021 /
Published online: 21 July 2021

Abstract
Over the past two decades, support vector machines (SVMs) have become a popular supervised machine learning model, and plenty of distinct algorithms are designed separately based on different KKT conditions of the SVM model for classification/regression with different losses, including convex and or nonconvex loss. In this paper, we propose an algorithm that can train different SVM models in a *unified* scheme. First, we introduce a definition of the least squares type of difference of convex loss (LS-DC) and show that the most commonly used losses in the SVM community are LS-DC loss or can be approximated by LS-DC loss. Based on the difference of convex algorithm (DCA), we then propose a unified algorithm called *UniSVM* which can solve the SVM model with any convex or nonconvex LS-DC loss, wherein only a vector is computed by the specifically chosen loss. UniSVM has a dominant advantage over all existing algorithms for training robust SVM models with nonconvex losses because it has a closed-form solution per iteration, while the existing algorithms always need to solve an L1SVM/L2SVM per iteration. Furthermore, by the low-rank approximation of the kernel matrix, UniSVM can solve large-scale nonlinear problems efficiently. To verify the efficacy and feasibility of the proposed algorithm, we perform many experiments on small artificial problems and large benchmark tasks both with and without outliers for classification and regression for comparison with state-of-the-art algorithms. The experimental results demonstrate that UniSVM can achieve comparable performance in less training time. The foremost advantage of UniSVM is that its core code in Matlab is less than 10 lines; hence, it can be easily grasped by users or researchers.

Keywords SVM (support vector machine) · DC programming · DCA (difference of convex algorithm) · LS-DC loss · low-rank approximation

This work was supported by the National Natural Science Foundation of China under Grant No. 61772020.

Editors: Daniel Fremont, Mykel Kochenderfer, Alessio Lomuscio, Dragos Margineantu, Cheng Soon-Ong.

✉ Shuisheng Zhou
sszhou@mail.xidian.edu.cn

[1] School of Mathematics and Statistics, Xidian University, Xi'An, China 726100

[2] School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China 100867

1 Introduction

Over the past two decades, support vector machines (SVMs) (Vapnik 1999, 2000), which are based on structural risk minimization, have become a computationally powerful machine learning method for supervised learning. They are widely used in classification and regression tasks (Vapnik 2000; Schölkopf and Smola 2002; Steinwart and Christmann 2008), such as disease diagnosis, face recognition, and image classification, etc.

Assuming that a training data set $\mathbb{T} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ is drawn independently and identically from a probability distribution on $(\mathcal{X}, \mathcal{Y})$ with $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} = \{-1, +1\}$ for classification or $\mathcal{Y} = \mathbb{R}$ for regression, the SVM model solves the following optimization problem:

$$\boldsymbol{w}^* \in \arg\min_{\boldsymbol{w}\in\mathbb{H}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{i=1}^m \ell(y_i, \langle \boldsymbol{w}, \phi(\boldsymbol{x}_i)\rangle), \tag{1}$$

where $\mathbb{H}$ is a reproducing kernel Hilbert space (RKHS) induced by a kernel function $\kappa(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z})\rangle$ with a feature mapping $\phi : \mathbb{R}^d \mapsto \mathbb{H}$, $\ell(\cdot, \cdot)$ is a margin-based loss with different choices, and λ is the regularizer. The output prediction function f is parameterized by $\boldsymbol{w}$ as $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \phi(\boldsymbol{x})\rangle$. Herein, we take the form without offset for f as in previous papers (Steinwart 2003; Keerthi et al. 2006; Steinwart et al. 2011). The offset can also be considered by adding an extra attribute 1 to every sample $\boldsymbol{x}$ or to its feature mapping $\phi(\boldsymbol{x})$.

For nonlinear problems, the model (1) cannot be solved efficiently because $\phi(\cdot)$ is always a high-dimensional mapping and possibly infinite. By applying the representer theorem (Schölkopf et al. 2001; Schölkopf and Smola 2002; Steinwart and Christmann 2008; Shalev-Shwartz and Ben-David 2014), there exists a vector $\boldsymbol{\alpha}^* \in \mathbb{R}^m$ such that the solution of (1) admits $\boldsymbol{w}^* = \sum_{i=1}^m \alpha_i^* \phi(\boldsymbol{x}_i)$. Hence, substituting $\boldsymbol{w} = \sum_{i=1}^m \alpha_i \phi(\boldsymbol{x}_i)$ in (1), we have the following equivalent finite dimensional optimization problem,

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^m} \frac{\lambda}{2}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha} + \frac{1}{m}\sum_{i=1}^m \ell(y_i, \boldsymbol{K}_i\boldsymbol{\alpha}), \tag{2}$$

where the kernel matrix $\boldsymbol{K}$ satisfies $\boldsymbol{K}_{i,j} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\boldsymbol{K}_i$ is the k-th row of $\boldsymbol{K}$. The similar model of (2) can also be derived by duality (Vapnik 2000; Boyd and Vandenberghe 2009), where the coefficients $\boldsymbol{\alpha}$ may be properly bounded (see (5), (6) and (9) for details).

Many scholars have studied different SVM models based on different loss functions. The typical works (Vapnik 2000; Suykens and Vandewalle 1999a; Keerthi et al. 2006; Zhou et al. 2009; Steinwart et al. 2011; Zhou 2013; Zhou et al. 2013; Zhou 2016) are focused on SVM models with convex loss, such as L1-SVM with the hinge loss, L2-SVM with the squared hinge loss, LS-SVM with the least squares least loss, and the support vector regression (SVR) with ε-insensitive loss, etc. The state-of-the-art SVM tool, *LibSVM* (including SVC and SVR) (Chang and Lin 2011), covers some cases with convex losses and has numerous applications.

The algorithms based on convex losses are sensitive to outliers, where "outlier" refers to the contaminated samples far away from the majority of instances with the same labels (Hampel et al. 2011) that may emerge through mislabeling. This is because the contaminated data have the largest weights (support values) to represent the output function in this case.

Many researchers have used nonconvex loss functions to weaken the influence of outliers. For example, Shen et al. (2003), Collobert et al. (2006), and Wu and Liu (2007) study the robust SVM with the truncated hinge loss; Tao et al. (2018) study the robust SVM with the truncated hinge loss and the truncated squared hinge loss. Based on difference of convex algorithm (DCA) procedure (Le Thi and Pham Dinh 2018; Yuille and Rangarajan 2003), researchers have presented algorithms to *iteratively* solve L1SVM/L2SVM to obtain the solutions of their proposed nonconvex models. By introducing the smooth nonconvex losses, Feng et al. (2016) propose the robust SVM models which solve a re-weighted L2SVM many times. See Subsect. 2.2.2 for the representative robust SVMs.

All the robust SVM algorithms mentioned above have double-layer loops. The inner loop is used to solve a convex problem with parameters adjustable by the outer loop, and the outer loop adjusts those parameters to reach the solution of the nonconvex model.

However, the inner loop of these algorithms is computationally expensive. For example, Collobert et al. (2006), Wu and Liu (2007), Feng et al. (2016), Tao et al. (2018) solve a constrained quadratic programming (QP) defined by L1SVM, L2SVM or re-weighted L2SVM, and all state-of-the-art methods for quadratic programming require substantial numbers of iterations (such as SMO (Platt 1999; Keerthi et al. 2001; Chen et al. 2006) or the tools `quadprog` in Matlab). In Tao et al. (2018), some efficient techniques based on the coordinate descent are given to reduce the cost of the inner loop, but it remains necessary to solve L1SVM/L2SVM, perhaps with smaller size.

There are three weaknesses to the existing algorithms of the robust SVM models. The first is that the total computational complexity is high, resulting in long training time which limits the algorithms in processing large-scale problems. The second is that most of the existing algorithms are only suitable for classification problems and require complicated modifications when applied for regression problems. The third is that all the existing algorithms are designed separately based on the special kinds of losses, thus costing much effort for the readers/users to learn the different algorithms or to change the losses before making use of them.

Recently, Chen and Zhou (2018) proposed the robust LSSVM based on the truncated least squares loss, which partly resolves the first two weaknesses by removing the inner loop and solving classification/regression tasks similarly). To extend this benefit to all the other losses, by defining an *LS-DC loss*, we propose a unified solution for different models with different losses, named as *UniSVM*, which overcomes all three aforementioned weaknesses.

Here, we only focus on the positive kernel case, namely, where $\mathbb{H}$ is an RKHS. For the nonpositive kernel case, Ong et al. (2004) generalized this type of learning problem to reproducing kernel Kreĭn spaces (RKKS) and verified that the representer theorem still holds in RKKS even if its regularization part is nonconvex. Recently, Xu et al. (2017) decomposed the regularization part as DC form and proposed an efficient learning algorithm where the loss is chosen as only the squared hinge loss. Our results in this work can be seamlessly generalized to the nonpositive case by the methods in Xu et al. (2017).

Our contributions in this work can be summarized as follows:

- We define a kind of loss with a DC decomposition, called *LS-DC loss*, and show that all the commonly used losses are LS-DC loss or can be approximated by LS-DC loss.
- We propose a UniSVM algorithm which can deal with any LS-DC loss in a unified scheme, including both convex and nonconvex losses and both classification and

regression losses, in which only one vector is dominated by the specifically chosen loss. Hence, it can train classification problems and regression problems in the same scheme.
- The proposed UniSVM has low computational complexity, even for nonconvex models, because it solves a system of linear equations iteratively, which has a closed-form solution. Hence, the inner loop disappears.
- By the efficient low-rank approximation of the kernel matrix, UniSVM can solve large-scale problems efficiently.
- In view of the theories of DCA, UniSVM converges to the globally optimal solution of the convex model, or to a critical point of the nonconvex model.
- UniSVM can be easily grasped by users or researchers because its core code in Matlab is less than 10 lines.

The notations in this paper are as follows. All the vectors and matrices are in bold style, such as $\boldsymbol{v}, \boldsymbol{x}_i$ or $\boldsymbol{K}$, and the set or space is noted as $\mathbb{M}, \mathbb{B}, \mathbb{R}^m$, etc. The scalar v_i is the i-th element of $\boldsymbol{v}$, the row vector $\boldsymbol{K}_i$ is the i-th row of $\boldsymbol{K}$, and $\boldsymbol{K}_{\mathbb{B}}$ is the submatrix of $\boldsymbol{K}$ with all *rows* in the index set $\mathbb{B}$. The transpose of the vector $\boldsymbol{v}$ or matrix $\boldsymbol{K}$ is noted as $\boldsymbol{v}^\top$ or $\boldsymbol{K}^\top$. $\boldsymbol{I}$ is an identity matrix with proper dimensions, $t_+ := \max\{t, 0\}$, and $\mathbb{1}_a = 1$ if the event a is true, and otherwise 0.

The rest of the paper is organized as follows. In Sect. 2 we review the DCA procedure and the related SVM models. In Sect. 3 we define an LS-DC loss which has a desirable DC decomposition and reveal its properties. In Sect. 4 we propose a UniSVM algorithm that can train the SVM model with any different LS-DC loss based on DCA. In Sect. 5 we verify the effectiveness of UniSVM with numerous experiments, and Sect. 6 concludes the paper.

2 Reviews of related works

We review the DCA procedure and some SVM models with convex and nonconvex loss in this section.

2.1 DC programming and DCA

DCA is an efficient nonconvex optimization technique, first introduced in Pham Dinh and El-Bernoussi (1986) and recently reviewed in Le Thi and Pham Dinh (2018), and has been successfully applied in machine learning (Yuille and Rangarajan 2003; Neumann et al. 2004; Collobert et al. 2006; Le Thi et al. 2008, 2009; Ong and Le Thi 2013; Xu et al. 2017; Tao et al. 2018; Chen and Zhou 2018). A function $F(x)$ is called a difference of convex (DC) function if $F(\boldsymbol{x}) = H(\boldsymbol{x}) - G(\boldsymbol{x})$, where $H(\boldsymbol{x})$ and $G(\boldsymbol{x})$ are convex functions. DC programming is used to solve

$$\min_{\boldsymbol{x}\in\mathcal{X}} F(\boldsymbol{x}) := H(\boldsymbol{x}) - G(\boldsymbol{x}), \tag{3}$$

where $H(\boldsymbol{x})$ and $G(\boldsymbol{x})$ are convex functions and $\mathcal{X}$ is a convex set.

DCA is a majorization-minimization algorithm (Naderi et al. 2019) which works by optimizing a sequence of upper-bounded convex functions of $F(\boldsymbol{x})$. For the current approximated solution $\boldsymbol{x}^k$, since $G(\boldsymbol{x}) \geq G(\boldsymbol{x}^k) + \langle \boldsymbol{x} - \boldsymbol{x}^k, \boldsymbol{v}^k \rangle$ with $\boldsymbol{v}^k \in \partial G(\boldsymbol{x}^k)$,

$H(\boldsymbol{x}) - \langle \boldsymbol{v}^k, \boldsymbol{x} - \boldsymbol{x}^k \rangle - G(\boldsymbol{x}^k)$ is an upper-bounded convex function of $F(\boldsymbol{x})$. Thus, to solve the DC problem (3), DCA iteratively obtains a new solution $\boldsymbol{x}^{k+1}$ by solving the convex programming as follows.

$$\boldsymbol{x}^{k+1} \in \arg\min_{\boldsymbol{x}} H(\boldsymbol{x}) - \langle \boldsymbol{v}^k, \boldsymbol{x} \rangle. \tag{4}$$

There is a convergence guarantee (Le Thi and Pham Dinh 2018). Particularly, if $H(\boldsymbol{x})$ is a quadratic function, the optimal problem (4) has a closed form solution. Thus, the DCA procedure has no inner iterations. This motivates us to design a DC decomposition for the losses of SVMs models to speed up the algorithm in Sect. 3.

2.2 SVM models with convex losses and nonconvex losses

2.2.1 SVM models with convex losses

If the hinge loss $\ell(y,t) := \max\{0, 1-yt\}$ is chosen in (1) for classification, then L1SVM is obtained by duality through the following expression (Vapnik 2000, 1999; Keerthi et al. 2006; Steinwart et al. 2011; Zhou 2013):

$$\text{L1SVM: } \min_{0 \le \boldsymbol{\beta} \le \frac{1}{m}} \frac{1}{2\lambda} \boldsymbol{\beta}^\top \widetilde{\boldsymbol{K}} \boldsymbol{\beta} - \boldsymbol{e}^\top \boldsymbol{\beta}, \tag{5}$$

where $\widetilde{\boldsymbol{K}}_{i,j} = y_i y_j \boldsymbol{K}_{i,j}$ and $\boldsymbol{e} = (1, \cdots, 1)^\top \in \mathbb{R}^m$. If the squared hinge loss $\ell(y,t) := \frac{1}{2}\max\{0, 1-yt\}^2$ is chosen in (1), then L2SVM is obtained by duality through the following expression (Vapnik 2000, 1999; Steinwart et al. 2011; Zhou 2013; Zhou et al. 2013, 2009):

$$\text{L2SVM: } \min_{0 \le \boldsymbol{\beta}} \frac{1}{2\lambda} \boldsymbol{\beta}^\top \left(\widetilde{\boldsymbol{K}} + \lambda m \boldsymbol{I} \right) \boldsymbol{\beta} - \boldsymbol{e}^\top \boldsymbol{\beta}, \tag{6}$$

where $\boldsymbol{I}$ is the identity matrix. With the solution $\boldsymbol{\beta}^*$, the unknown sample $\boldsymbol{x}$ is predicted as $sgn(f(\boldsymbol{x}))$ with $f(\boldsymbol{x}) = \frac{1}{\lambda} \sum_{i=1}^m y_i \beta_i^* \kappa(x, x_i)$ for model (5) or (6).

If the least squares loss $\ell(y,t) := \frac{1}{2}(1-yt)^2 = \frac{1}{2}(y-t)^2$ is chosen in (1), the LSSVM model obtained by duality (Suykens and Vandewalle 1999a, b; Suykens et al. 2002; Jiao et al. 2007) is

$$\text{LSSVM: } \min_{\boldsymbol{\beta}} \frac{1}{2\lambda} \boldsymbol{\beta}^\top (\boldsymbol{K} + \lambda m \boldsymbol{I}) \boldsymbol{\beta} - \boldsymbol{y}^\top \boldsymbol{\beta}, \tag{7}$$

with a unique nonsparse solution, where $\boldsymbol{y} = (y_1, \cdots, \boldsymbol{y}_m)^\top$. By choosing the least squares loss in (1), Zhou (2016) recently proposed the primal LSSVM (PLSSVM) based on the representer theorem[1] as

$$\text{PLSSVM: } \min_{\boldsymbol{\beta}} \frac{1}{2\lambda} \boldsymbol{\beta}^\top \left(m\lambda \boldsymbol{K} + \boldsymbol{K}\boldsymbol{K}^\top \right) \boldsymbol{\beta} - \boldsymbol{y}^\top \boldsymbol{K} \boldsymbol{\beta}, \tag{8}$$

[1] For consistency with the model induced by duality, we let $\boldsymbol{w} = \frac{1}{\lambda} \sum_{i=1}^m \boldsymbol{\beta}_i \phi(\boldsymbol{x}_i)$ here.

which may have a sparse solution if $\boldsymbol{K}$ has low rank or can be approximated as a low rank matrix. With the solution $\boldsymbol{\beta}^*$, the unknown sample $\boldsymbol{x}$ is predicted as $sgn(f(\boldsymbol{x}))$(classification) or $f(\boldsymbol{x})$ (regression) with $f(\boldsymbol{x}) = \frac{1}{\lambda}\sum_{i=1}^m \beta_i^* \kappa(x, x_i)$ for model (7) or (8).

If the ε-insensitive loss $\ell_\varepsilon(y,t) := (|y-t|-\varepsilon)_+$ is chosen in (1) for the regression problem, then SVR is obtained as

$$\text{SVR:} \min_{0\le\boldsymbol{\beta},\hat{\boldsymbol{\beta}}\le\frac{1}{m}} \frac{1}{2\lambda}(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}})^\top \boldsymbol{K}(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}}) + \varepsilon\sum_{i=1}^m(\boldsymbol{\beta}+\hat{\boldsymbol{\beta}}) + \sum_{i=1}^m y_i(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}}), \tag{9}$$

and with the solution $(\boldsymbol{\beta}^*, \hat{\boldsymbol{\beta}}^*)$, the prediction of the new input $\boldsymbol{x}$ is $f(\boldsymbol{x}) = \frac{1}{\lambda}\sum_{i=1}^m(\beta_i^* - \hat{\beta}_i^*)\kappa(x, x_i)$.

2.2.2 Robust SVM models with nonconvex losses

To improve the robustness of the L1SVM model (5), in Collobert et al. (2006) the hinge loss $(1-yt)_+$ in (1) is truncated as the ramp loss $\min\{(1-yt)_+, a\}$ with $a>0$ and decomposed into DC form $(1-yt)_+ - (1-yt-a)_+$. The problem (1) is posed as a DC programming:

$$\min_{\boldsymbol{w}\in\mathbb{H}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{i=1}^m(1-y_i\langle\boldsymbol{w},\phi(\boldsymbol{x}_i)\rangle)_+ - \frac{1}{m}\sum_{i=1}^m(1-y_i\langle\boldsymbol{w},\phi(\boldsymbol{x}_i)\rangle - a)_+.$$

Then, based on the DC procedure and Lagrange duality, a robust L1SVM is proposed by iteratively solving the following L1SVM problem at the $(k+1)$−th iteration.

$$\boldsymbol{\beta}^{k+1} \in \arg\min_{0\le\boldsymbol{\beta}\le\frac{1}{m}} \frac{1}{2\lambda}(\boldsymbol{\beta}-\boldsymbol{v}^k)^\top\widetilde{\boldsymbol{K}}(\boldsymbol{\beta}-\boldsymbol{v}^k) - \boldsymbol{e}^\top\boldsymbol{\beta}, \tag{10}$$

where $\boldsymbol{v}^k$ satisfies $v_i^k = \frac{1}{m}\cdot\mathbb{1}_{1-\widetilde{\boldsymbol{K}}_i(\boldsymbol{\beta}^k-\boldsymbol{v}^k)/\lambda>a}, i=1,\cdots,m$. Similar analysis with a different form appears in Wu and Liu (2007) and Tao et al. (2018).

To improve the robustness of L2SVM (6), Tao et al. (2018) truncated the squared hinge loss $(1-yt)_+^2$ as $\min\{(1-yt)_+^2, a\}$ in (1) and decomposed into DC form $(1-yt)_+^2 - \left((1-yt)_+^2 - a\right)_+$. Based on DCA, the robust solution of L2SVM is given by iteratively solving

$$\boldsymbol{\beta}^{k+1} \in \arg\min_{\boldsymbol{\beta}\ge 0} \frac{1}{2\lambda}(\boldsymbol{\beta}-\boldsymbol{v}^k)^\top\widetilde{\boldsymbol{K}}(\boldsymbol{\beta}-\boldsymbol{v}^k) + \frac{m}{2}\boldsymbol{\beta}^\top\boldsymbol{\beta} - \boldsymbol{e}^\top\boldsymbol{\beta}, \tag{11}$$

where $\boldsymbol{v}^k$ satisfies $v_i^k = \frac{1}{m}(1-\frac{1}{\lambda}\widetilde{\boldsymbol{K}}_i(\boldsymbol{\beta}^k-\boldsymbol{v}^k))\cdot\mathbb{1}_{1-\widetilde{\boldsymbol{K}}_i(\boldsymbol{\beta}^k-\boldsymbol{v}^k)/\lambda>\sqrt{a}}, i=1,\cdots,m$. However, the analysis in Tao et al. (2018) pointed out that (11) is not a "satisfactory learning algorithm" in this case. By deleting the current outliers, which satisfy $1-y_if(\boldsymbol{x}_i) > \sqrt{a}$ from the training set iteratively, Tao et al. (2018) proposed a multistage SVM (MS-SVM) to solve the robust L2SVM. They first solve an original L2SVM (6) and then solve smaller L2SVMs iteratively.

Although Tao et al. (2018) propose improved methods using coordinate descent and an inexpensive scheme, all of the given algorithms still solve a constrained QP per iteration, and possibly with smaller size.

In contrast, Feng et al. (2016) propose a robust SVM model, in which the following smooth nonconvex loss

$$\ell_a(y,t) = a\Big(1 - \exp\Big(-\tfrac{1}{a}(1-yt)_+^2\Big)\Big) \tag{12}$$

with $a > 0$ is chosen in (1). The loss (12) is approximated as the squared hinge loss $(1-yt)_+^2$ when $a \to +\infty$ and can be considered as a smooth approximation of the truncated squared hinge loss $\min\{(1-yt)_+^2, a\}$. After analysis of the KKT conditions of the given model, Feng et al. (2016) proposed the algorithm by solving the following re-weighted L2SVM iteratively:

$$\boldsymbol{\beta}^{k+1} \in \arg\min_{\boldsymbol{\beta}\geq 0} \tfrac{1}{2\lambda}\boldsymbol{\beta}^\top\Big(\widetilde{\boldsymbol{K}} + m\lambda \boldsymbol{D}^k\Big)\boldsymbol{\beta} - \boldsymbol{e}^\top\boldsymbol{\beta} \tag{13}$$

where $\boldsymbol{D}^k$ is a diagonal matrix satisfying $\boldsymbol{D}^k_{i,i} = \Big(\psi'((1-\widetilde{\boldsymbol{K}}_i\boldsymbol{\beta}^k)_+^2)\Big)^{-1}$ with $\psi(u) = a\Big(1-\exp(-\tfrac{1}{a}u)\Big)$.

All of these algorithms for robust SVM models (Collobert et al. 2006; Wu and Liu 2007; Tao et al. 2018; Feng et al. 2016) must solve a constrained QP in the inner loops, which results in a long training time.

Based on the decomposition of the truncated *least squares loss* $\min\{(1-yt)^2, a\}$ in (1) or (2) as $(1-yt)^2 - ((1-yt)^2 - a)_+$ and the representer theorem, the robust sparse LSSVM (RSLSSVM) was studied in Chen and Zhou (2018) by solving

$$\boldsymbol{\alpha}^{k+1} \in \arg\min_{\boldsymbol{\alpha}\in\mathbb{R}^m} \frac{\lambda}{2}\boldsymbol{\alpha}^\top\boldsymbol{K}\boldsymbol{\alpha} + \tfrac{1}{2m}\sum_{i=1}^m (y_i - \boldsymbol{K}_i\boldsymbol{\alpha})^2 - \tfrac{1}{m}\langle \boldsymbol{K}\boldsymbol{v}^k, \boldsymbol{\alpha}\rangle, \tag{14}$$

with $\boldsymbol{v}^k$ as $v_i^k = -(y_i - \boldsymbol{K}_i\boldsymbol{\alpha}^k)\mathbb{1}_{|1-y_i\boldsymbol{K}_i\boldsymbol{\alpha}^k|>\sqrt{a}}$.

The model (14) for solving the nonconvex SVM has three advantages. The first is that it has a closed-form solution since it is an unconstrained QP. Second, it has a sparse solution if $\boldsymbol{K}$ has low rank or can be approximated by a low rank matrix (see (Zhou 2016) for details). Thus, it can be solved efficiently. Furthermore, (14) can also be applied to regression problems directly.

To extend those benefits to all the other losses for classification tasks and regression tasks simultaneously, we define LS-DC loss in Sect. 3 and propose a unified algorithm in Sect. 4, which includes all SVM models (including the classification/regression SVM with convex loss and nonconvex loss) in a unified scheme.

3 LS-DC loss function

Here, we first define a kind of loss called LS-DC loss, and then show that most popular losses are in fact LS-DC loss or can be approximated by LS-DC loss.

For any margin-based loss $\ell(y,t)$ of SVM, let $\psi(u)$ satisfy $\psi(1-yt) := \ell(y,t)$ for classification loss or $\psi(y-t) := \ell(y,t)$ for regression loss. To obtain a useful DC decomposition of the loss $\ell(y,t)$, we propose the following definition:

Definition 1 (LS-DC loss) We call $\ell(y,t)$ *a least squares type DC loss*, abbreviated as *LS-DC loss*, if there exists a constant A $(0 < A < +\infty)$ such that $\psi(u)$ has the following DC decomposition:

$$\psi(u) = Au^2 - (Au^2 - \psi(u)). \tag{15}$$

The essence of the definition demands that $Au^2 - \psi(u)$ be convex. The following theorem is clear:

Theorem 1 *If the loss $\psi(u)$ is second-order derivable and $\psi''(u) \leq M$, then it is an LS-DC loss with parameter $A \geq \frac{M}{2}$.*

Not all losses are LS-DC losses, even the convex losses. We will show that the hinge loss and the ε-insensitive loss are not LS-DC losses. However, they can be approximated by LS-DC losses.

Next, we will show that most losses used in the SVM community are LS-DC losses or can be approximated by LS-DC loss. The proofs are listed in Appendix A.

Proposition 1 (LS-DC property of classification losses) *The most commonly used classification losses are cases of LS-DC loss or can be approximated by LS-DC losses. We enumerate them as follows.*

(a) *The least squares loss $\ell(y,t) = (1-yt)^2$ is an LS-DC loss with $Au^2 - \psi(u)$ vanished.*
(b) *The truncated least squares loss $\ell(y,t) = \min\{(1-yt)^2, a\}$ is an LS-DC loss with $A \geq 1$.*
(c) *The squared hinge loss $\ell(y,t) = (1-yt)_+^2$ is an LS-DC loss with $A \geq 1$.*
(d) *The truncated squared hinge loss $\ell(y,t) = \min\{(1-yt)_+^2, a\}$ is an LS-DC loss with $A \geq 1$.*
(e) *The hinge loss $\ell(y,t) = (1-yt)_+$ is not an LS-DC loss. However, if it is approximated as $\frac{1}{p}\log(1+\exp(p(1-yt)))$ with a finite p, we obtain an LS-DC loss with $A \geq p/8$.*
(f) *The ramp loss $\ell(y,t) = \min\{(1-yt)_+, a\}$ is also not an LS-DC loss. However, we can give two smoothed approximations of the ramp loss*:

$$\ell_a(y,t) = \begin{cases} \frac{2}{a}(1-yt)_+^2, & 1-yt \leq \frac{a}{2}, \\ a - \frac{2}{a}(a-(1-yt))_+^2, & 1-yt > \frac{a}{2}. \end{cases} \tag{16}$$

$$\ell_{(a,p)}(y,t) = \frac{1}{p}\log\left(\frac{1+\exp(p(1-yt))}{1+\exp(p(1-yt-a))}\right). \tag{17}$$

The first one has the same support set as the ramp loss, and the second one is derivable with any order. The loss (16) is an LS-DC loss with $A \geq 2/a$ and (17) is an LS-DC loss with $A \geq p/8$.
(g) *The nonconvex smooth loss (12) proposed in Feng et al. (2016) is an LS-DC loss with $A \geq 1$.*
(h) *Following the nonconvex smooth loss (12), we generalize it as*

$$\ell_{(a,b,c)}(y,t) = a\left(1 - \exp\left(-\tfrac{1}{b}(1-yt)_+^c\right)\right), \tag{18}$$

where $a, b > 0, c \geq 2$. The loss (18) is an LS-DC loss with the parameter $A \geq \frac{1}{2}M(a,b,c)$, where

$$M(a,b,c) := \frac{ac}{b^{2/c}}\left((c-1)(h(c))^{1-2/c} - c(h(c))^{2-2/c}\right)e^{-h(c)}, \tag{19}$$
$$\text{with } h(c) := \left(3(c-1) - \sqrt{5c^2-6c+1}\right)/(2c).$$

Regarding the new proposed loss (18), we offer the following comments:

Remark 1 Two more parameters are introduced to the loss (18) to make it more flexible. The parameter a, which is the limitation of the loss function if $1 - yt \to \infty$, describes the effective value (or saturated value) of the loss function for large inputs. The parameter b, which characterizes the localization property of the loss functions, describes the rate of the loss function saturated to its maximum and minimum. By uncoupling a and b, we improve the flexibility of the robust loss. For example, the inflection point of (12) is $yt = 1 - \sqrt{a/2}$, which is directly controlled by the saturated value a, and the inflection point of (18) is $yt = 1 - \sqrt{b/2}$ if $c = 2$, which is only controlled by the parameter b. In experiments, by simply adjusting a, b, and c, we can obtain better performance.

The most commonly used losses for classification are summarized in Table 5 in Appendix B. Some classification losses and their LS-DC decompositions are also plotted in Fig. 1.

Proposition 2 (LS-DC property of regression losses) *The commonly used regression losses are LS-DC loss or can be approximated by LS-DC loss. We enumerate them as follows.*

(1) *The least squares loss and the truncated least squares loss are all LS-DC losses with* $A \geq 1$.

(2) *The* ε*-insensitive loss* $\ell_\varepsilon(y,t) := (|y-t| - \varepsilon)_+$, *mostly used for SVR, is not an LS-DC loss. However, we can smooth it as*

$$\ell_{(\varepsilon,p)}(y,t) := \frac{1}{p}\log(1+\exp(-p(y-t+\varepsilon))) + \frac{1}{p}\log(1+\exp(p(y-t-\varepsilon))), \tag{20}$$

which is LS-DC loss with $A \geq p/4$.

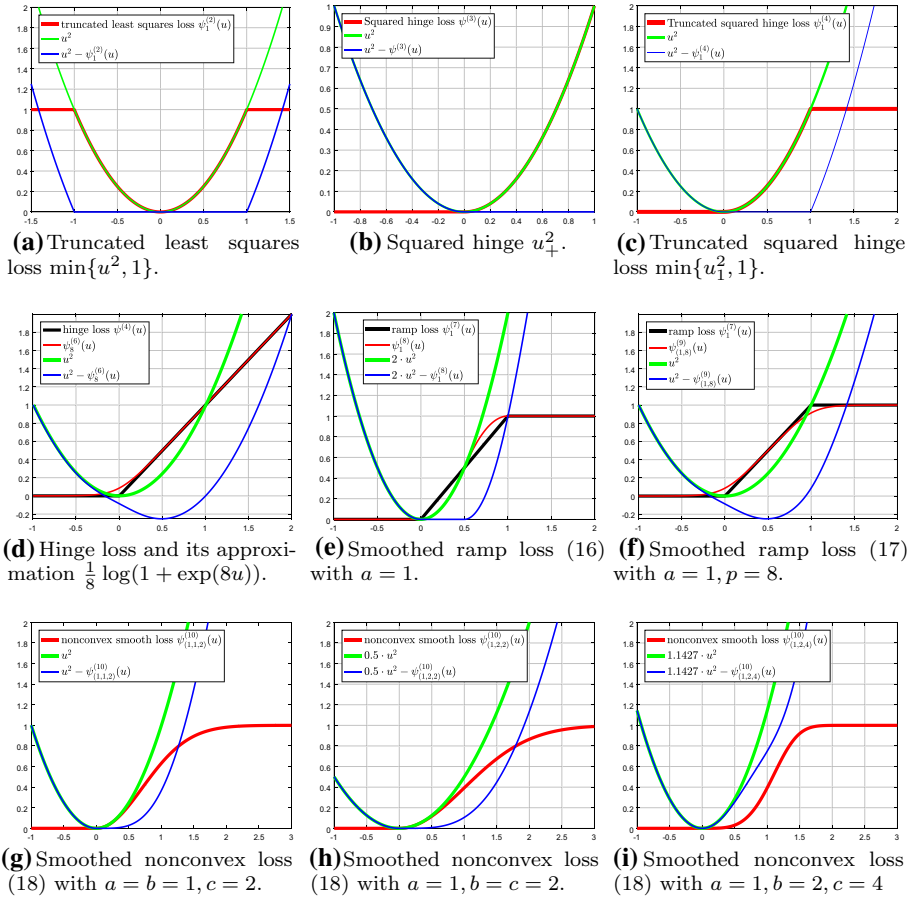
(3) *The absolute loss* $\ell(y,t) = |y-t|$ *is also not an LS-DC loss. However, it can be smoothed by LS-DC losses. For instance, we can use the Huber loss*

$$\ell_\delta(y,t) = \begin{cases} \frac{1}{2\delta}(y-t)^2, & |y-t| \leq \delta, \\ |y-t| - \frac{\delta}{2}, & |y-t| > \delta, \end{cases}$$

which approximates the absolute loss is an LS-DC loss with $A \geq 1/(2\delta)$; *Setting* $\varepsilon = 0$ *in* (20), $A \geq p/4$.

(4) *The truncated absolute loss* $\ell_a(y,t) := \min\{|y-t|, a\}$ *can be approximated by the truncated Huber loss* $\min\{\ell_\delta(y,t), a\}$, *which is an LS-DC loss with* $A \geq 1/(2\delta)$.

Some regression losses and their DC decompositions are plotted in Fig. 2.



(a) Truncated least squares loss $\min\{u^2, 1\}$. **(b)** Squared hinge u_+^2. **(c)** Truncated squared hinge loss $\min\{u_1^2, 1\}$.

(d) Hinge loss and its approximation $\frac{1}{8}\log(1+\exp(8u))$. **(e)** Smoothed ramp loss (16) with $a=1$. **(f)** Smoothed ramp loss (17) with $a=1, p=8$.

(g) Smoothed nonconvex loss (18) with $a=b=1, c=2$. **(h)** Smoothed nonconvex loss (18) with $a=1, b=c=2$. **(i)** Smoothed nonconvex loss (18) with $a=1, b=2, c=4$

Fig. 1 The plots of some LS-DC losses for classification and their DC decompositions: "**Red curve** = **Green curve**-**Blue curve**". In the plot, the **Black curve** (if it exists) is the plot of the original non-LS-DC loss which is approximated by an LS-DC loss (**Red curve**), the **Green curve** is the function $A\mu^2$ and the **Blue cruve** is the convex function $A\mu^2$-$\Psi(u)$. The loss names in the legends are defined in Table 5 in Appendix B. All of the LS-DC parameters **A** are chosen as the lower bounds in Table 5, and increasing the value of **A** can make the **Blue cruve** "smoother"

4 Unified algorithm for SVM models with LS-DC losses

Let $\ell(y,t)$ be any LS-DC loss discussed in Section 3, and let $\psi(u)$ satisfying $\psi(1-yt)=\ell(y,t)$ (for classification task) or $\psi(y-t)=\ell(y,t)$ (for regression tasks) have the DC decomposition (15) with parameter $A>0$. The SVM model (2) with any loss can then be decomposed as

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^m} \lambda\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha} + \frac{A}{m}\|\boldsymbol{y}-\boldsymbol{K}\boldsymbol{\alpha}\|^2 - \left(\frac{A}{m}\|\boldsymbol{y}-\boldsymbol{K}\boldsymbol{\alpha}\|^2 - \frac{1}{m}\sum_{i=1}^m \psi\left(r_i\right)\right), \tag{21}$$

where $r_i = 1-y_i\boldsymbol{K}_i\boldsymbol{\alpha}$ (for classification) and $r_i = y_i - \boldsymbol{K}_i\boldsymbol{\alpha}$ (for regression).

Using the DCA procedure (4) with an initial point $\boldsymbol{\alpha}^0$, a stationary point of (21) can be iteratively reached by solving

$$\boldsymbol{\alpha}^{k+1} \in \arg\min_{\boldsymbol{\alpha}\in\mathbb{R}^m} \lambda\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha} + \frac{A}{m}\|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|^2 + \left\langle \frac{2}{m}\boldsymbol{K}\big(A(\boldsymbol{y} - \boldsymbol{\xi}^k) - \boldsymbol{\gamma}^k\big), \boldsymbol{\alpha}\right\rangle, \tag{22}$$

where $\boldsymbol{\xi}^k = \boldsymbol{K}\boldsymbol{\alpha}^k$ and $\boldsymbol{\gamma}^k = (\gamma_1^k, \gamma_2^k, \cdots, \gamma_m^k)^\top$ satisfies

$$\gamma_i^k \in \tfrac{1}{2} y_i \partial\psi(1 - y_i\xi_i^k)(\text{classication}) \text{ or } \gamma_i^k \in \tfrac{1}{2}\partial\psi(y_i - \xi_i^k)(\text{regression}), \tag{23}$$

where $\partial\psi(u)$ indicates the subdifferential of the convex function $\psi(u)$. The related losses and their derivatives or subdifferentials for updating $\boldsymbol{\gamma}^k$ in (23) are listed in Table 5 in Appendix B.

The KKT conditions of (22) are

$$\left(\frac{\lambda m}{A}\boldsymbol{K} + \boldsymbol{K}\boldsymbol{K}^\top\right)\boldsymbol{\alpha} = \boldsymbol{K}(\boldsymbol{\xi}^k - \frac{1}{A}\boldsymbol{\gamma}^k). \tag{24}$$

By solving (24), we propose a unified algorithm that can train SVM models with any LS-DC loss. For different LS-DC losses (either classification loss or regression loss), we just need to calculate the different $\boldsymbol{\gamma}$ by (23). We refer to the algorithm as **UniSVM**, which is summarized as Algorithm 1.

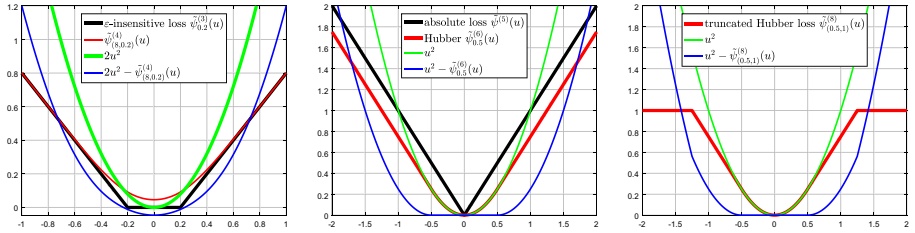
Algorithm 1 UniSVM(Unified SVM)

Input: Given a training set $\mathbb{T} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$ or $y_i \in \mathbb{R}$; Kernel matrix $\boldsymbol{K}$ satisfying $\boldsymbol{K}_{i,j} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, or $\boldsymbol{P}$ satisfying $\boldsymbol{P}\boldsymbol{P}^\top \approx \boldsymbol{K}$ satisfying $\boldsymbol{P}_{\mathbb{B}}\boldsymbol{P}^\top = \boldsymbol{K}_{\mathbb{B}}$; Any LS-DC loss function $\psi(u)$ with parameter $A > 0$; The regularizer λ.
Output: The prediction function $f(\boldsymbol{x}) = \sum_{i=1}^m \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{x})$ with $\boldsymbol{\alpha} = \boldsymbol{\alpha}^k$.
1: $\boldsymbol{\gamma}^0 = 0$, $\boldsymbol{\xi}^0 = \boldsymbol{y}$; Set $k := 0$.
2: **while** not convergence **do**
3: Solving (24) with respect to (25), (26) or (28) to obtain $\boldsymbol{\alpha}^{k+1}$, where the inversion is only calculated in the first iteration;
4: Update $\boldsymbol{\xi}^{k+1} = \boldsymbol{K}\boldsymbol{\alpha}^{k+1}$ or $\boldsymbol{\xi}^{k+1} = \boldsymbol{P}\boldsymbol{P}_{\mathbb{B}}^\top\boldsymbol{\alpha}_{\mathbb{B}}^{k+1}$, $\boldsymbol{\gamma}^{k+1}$ by (23); $k := k + 1$.
5: **end while**

The new algorithm possesses the following advantages:

- It is suitable for training any kind of SVM models with any LS-DC losses, including convex loss and nonconvex loss. The training process for classification problems is also the same as for regression problems. The proposed UniSVM is therefore a truly unified algorithm.
- Unlike the existing algorithms for nonconvex loss (Collobert et al. 2006; Wu and Liu 2007; Tao et al. 2018; Feng et al. 2016) that must iteratively solve L1SVM/L2SVM or reweighted L2SVM in the inner loops, UniSVM is free of the inner loop because it solves a system of linear equations (24) with a closed-form solution per iteration.
- According to the studies on LSSVM in Zhou (2016), the problem (24) may have multiple solutions, including some sparse solutions, if $\boldsymbol{K}$ has low rank[2]. This is of vital importance for training large-scale problems efficiently. Details will be discussed in Subsect. 4.2.

[2] $\boldsymbol{K}$ is always low rank in computing, for there are always many similar samples in the training set, causing the corresponding columns of the kernel matrix to be (nearly) linearly dependent.



(a) Smoothed ε-insensitive loss **(b)** Absolute and Hubber loss **(c)** Truncated Hubber loss

Fig. 2 The plots of some LS-DC losses for regression and their DC decompositions: "**Red curve = Green curve-Blue curve**". In the plot, the **Black curve** (if it exists) is the plot of the original non-LS-DC loss which is approximated by an LS-DC loss (**Red curve**), the **Green curve** is the function $A\mu^2$ and the Blue curve is the convex function $A\mu^2$-$\Psi(u)$. The loss names in the legends are defined in Table 5 in Appendix B. All of the LS-DC parameters A are chosen as the lower bounds in Table 5, and increasing the value of A can make the Blue curve "smoother"

- In experiments, we always set $\boldsymbol{\xi}^0 = \boldsymbol{y}$ and $\boldsymbol{\gamma}^0 = 0$ instead of giving an $\boldsymbol{\alpha}^0$ to begin the algorithm. This is equivalent to starting the algorithm from the solution of LSSVM, which is a moderate guess of the initial point, even for nonconvex loss.

In Subsect. 4.1, we present an easily grasped version of the proposed *UniSVM* in the case that the full kernel $\boldsymbol{K}$ is available. In Subsect. 4.2, we propose an efficient method to solve the KKT conditions (24) for *UniSVM* even if the full kernel matrix is unavailable. The Matlab code is also given in Appendix C.

4.1 Solving UniSVM with full kernel matrix available

If the full kernel matrix $\boldsymbol{K}$ is available and $\frac{\lambda m}{A}\boldsymbol{I} + \boldsymbol{K}$ can be inverted cheaply, we note that $\boldsymbol{Q} = \left(\frac{\lambda m}{A}\boldsymbol{I} + \boldsymbol{K}\right)^{-1}$ and can prove that

$$\boldsymbol{\alpha}^{k+1} = \boldsymbol{Q}(\boldsymbol{\xi}^k - \tfrac{1}{A}\boldsymbol{\gamma}^k) \tag{25}$$

is one nonsparse solution of (24). It should be noted that $\boldsymbol{Q}$ is only calculated once. Hence, after the first iteration, $\boldsymbol{\alpha}^{k+1}$ will be reached within $O(m^2)$.

Furthermore, if $\boldsymbol{K}$ is low rank and can be factorized as $\boldsymbol{K} = \boldsymbol{P}\boldsymbol{P}^\top$ with a full-column rank $\boldsymbol{P} \in \mathbb{R}^{m\times r}$ $(r < m)$, the cost of the process can be reduced through two methods. One is SMW identity (Golub and Loan 1996), which determines the cost $O(mr^2)$ to compute $\boldsymbol{P}^\top\boldsymbol{P}$, the cost $O(r^3)$ to obtain the inversion $\hat{\boldsymbol{Q}} = \left(\frac{\lambda m}{A}\boldsymbol{I} + \boldsymbol{P}^\top\boldsymbol{P}\right)^{-1} \in \mathbb{R}^{r\times r}$ once, and the cost within $O(mr)$ to update the nonsparse $\boldsymbol{\alpha}^{k+1}$ per iteration as

$$\boldsymbol{\alpha}^{k+1} = \tfrac{A}{\lambda m}\left(\boldsymbol{I} - \boldsymbol{P}\hat{\boldsymbol{Q}}\boldsymbol{P}^\top\right)(\boldsymbol{\xi}^k - \tfrac{1}{A}\boldsymbol{\gamma}^k). \tag{26}$$

The other is the method employed in subsection 4.2 to obtain a sparse solution of (24).

4.2 Solving UniSVM for large-scale training with a sparse solution

For large-scale problems, the full kernel matrix $\boldsymbol{K}$ is always unavailable because of the limited memory and the computational complexity. Hence, we should obtain the sparse solution of the model, since in this case $\boldsymbol{K}$ is always low rank or can be approximated by a low-rank matrix.

To obtain the low-rank approximation of $\boldsymbol{K}$, we can use the Nyström approximation (Sun et al. 2015), which is a random sampling method, or the pivoted Cholesky factorization method proposed in Zhou (2016) that has a guarantee to minimize the trace norm of the approximation error greedily. The gaining approximation of $\boldsymbol{K}$ is $\boldsymbol{P}\boldsymbol{P}^\top$, where $\boldsymbol{P} = [\boldsymbol{P}_{\mathbb{B}}^\top \ \boldsymbol{P}_{\mathbb{N}}^\top]^\top$ is a full column rank matrix with $\boldsymbol{P}_{\mathbb{B}} \in \mathbb{R}^{r\times r}$ ($r \ll m$) and $\mathbb{B} \subset \{1, 2, \cdots, m\}$ is the index set corresponding to the only visited r columns of $\boldsymbol{K}$. Both algorithms satisfy the condition that the total computational complexity is within $O(mr^2)$, and $\boldsymbol{K}_{\mathbb{B}}$—the visited rows of $\boldsymbol{K}$ corresponding to set $\mathbb{B}$—can be reproduced exactly as $\boldsymbol{P}_{\mathbb{B}}\boldsymbol{P}^\top$.

Replacing $\boldsymbol{K}$ with $\boldsymbol{P}\boldsymbol{P}^\top$ in (24), we have

$$\boldsymbol{P}(\tfrac{\lambda m}{A}\boldsymbol{I} + \boldsymbol{P}^\top\boldsymbol{P})\boldsymbol{P}^\top\boldsymbol{\alpha} = \boldsymbol{P}(\boldsymbol{P}^\top(\boldsymbol{\xi}^k - \tfrac{1}{A}\boldsymbol{\gamma}^k)),$$

which can be simplified as

$$\left(\tfrac{\lambda m}{A}\boldsymbol{I} + \boldsymbol{P}^\top\boldsymbol{P}\right)\boldsymbol{P}^\top\boldsymbol{\alpha} = \boldsymbol{P}^\top(\boldsymbol{\xi}^k - \tfrac{1}{A}\boldsymbol{\gamma}^k). \tag{27}$$

This is because $\boldsymbol{P}$ is a full column rank matrix. By simple linear algebra, if we let $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_{\mathbb{B}}^\top \ \boldsymbol{\alpha}_{\mathbb{N}}^\top]^\top$ be a partition of $\boldsymbol{\alpha}$ corresponding to the partition of $\boldsymbol{P}$, then we can set $\boldsymbol{\alpha}_{\mathbb{N}} = 0$ to solve (27). Thus, (27) is equivalent to

$$\left(\tfrac{\lambda m}{A}\boldsymbol{I} + \boldsymbol{P}^\top\boldsymbol{P}\right)\boldsymbol{P}_{\mathbb{B}}^\top\boldsymbol{\alpha}_{\mathbb{B}} = \boldsymbol{P}^\top(\boldsymbol{\xi}^k - \tfrac{1}{A}\boldsymbol{\gamma}^k).$$

We then have

$$\boldsymbol{\alpha}_{\mathbb{B}}^{k+1} = \overline{\boldsymbol{Q}}\boldsymbol{P}^\top(\boldsymbol{\xi}^k - \tfrac{1}{A}\boldsymbol{\gamma}^k). \tag{28}$$

where $\overline{\boldsymbol{Q}} = \left((\tfrac{\lambda m}{A}\boldsymbol{I} + \boldsymbol{P}^\top\boldsymbol{P})\boldsymbol{P}_{\mathbb{B}}^\top\right)^{-1}$, $\boldsymbol{\xi}^k = \boldsymbol{P}\boldsymbol{P}_{\mathbb{B}}^\top\boldsymbol{\alpha}_{\mathbb{B}}^k$, and $\boldsymbol{\gamma}^k$ is updated by (23).

Notice that $\overline{\boldsymbol{Q}}$ is only calculated in the first iteration with the cost $O(r^3)$. The cost of the algorithm is $O(mr^2)$ for the first iteration, and $O(mr)$ for the following iterations. Hence, UniSVM can be run very efficiently.

5 Experimental studies

In this section, we present experimental results to illustrate the effectiveness of the proposed unified model. All the experiments are run on a computer with an Intel Core i5-6500 CPU @3.20GHz×4 and a maximum memory of 8GB for all processes; the computer runs Windows 7 with Matlab R2016b. The comparators include L1SVM and SVR solved by `LibSVM`, L2SVM and the robust SVM modes in Collobert et al. (2006), Tao et al. (2018), Feng et al. (2016).

5.1 Intuitive comparison of UniSVM with other SVM models on small data sets

In this subsection, we first present experiments to show that the proposed UniSVM with convex loss can obtain a comparable performance by solving L1SVM, L2SVM and SVR on the small data sets. Second, we perform experiments to illustrate that UniSVM with nonconvex loss more efficiently obtains comparable performance to the algorithms in Collobert et al. (2006), Tao et al. (2018), and Feng et al. (2016) by solving robust SVMs with nonconvex loss. We have implemented UniSVM in two cases; one is in (25) with the full kernel matrix $\boldsymbol{K}$ available, called *UniSVM-full*, and the other obtains the sparse solution of the model by (28), where $\boldsymbol{K}$ is approximated as $\boldsymbol{P}\boldsymbol{P}^{\top}$ with $\boldsymbol{P} \in \mathbb{R}^{m\times r}(r \ll m)$, noted as *UniSVM-app*. The latter has the potential to resolve large-scale tasks. L1SVM and SVR are solved by the efficient tools `LibSVM` (Chang and Lin 2011), and the other related models (L2SVM, the robust L1SVM and the robust L2SVM) are solved by the solver of quadratic programming `quadprog.m` in Matlab.

5.1.1 On convex loss cases

The first experiment is a hard classification task on the highly nonlinearly separable "xor" dataset shown in Fig. 3, where the instances are generated by uniform sampling with 400 training samples and 400 test samples. The kernel function is $\kappa(\boldsymbol{x},\boldsymbol{z}) = \exp(-\gamma\|\boldsymbol{x}-\boldsymbol{z}\|^2)$ with $\gamma = 2^{-1}$, λ is set as 10^{-5} and $r = 10$ for UniSVM-app. The experimental results are plotted in Fig. 3 and the detailed information of the experiments is given as the captions and the subtitles of the figures.

From the experimental results in Fig. 3, we have the following findings:

- The proposed UniSVM for solving L1SVM or L2SVM can obtain similar performance compared with the state-of-the-art algorithms (LibSVM/quadprog). Of course, in those smaller cases, LibSVM is more efficient than UniSVM, since they are only designed for SVM with convex losses.
- The low-rank approximation of the kernel matrix can significantly accelerate UniSVM, and the acceleration rate is approximated as $\frac{m}{r}$.
- The red curves in Fig. 3c and f reveal that UniSVM is the majorization-minimization algorithm (Naderi et al. 2019). All cases of UniSVM reach the optimal value of L1SVM or L2SVM from above. In this setting, if $r > 15$, the difference of the objective values between UniSVM-full and UniSVM-app will vanish.

Thus, the advantage of UniSVM lies in solving the large-scale tasks with low-rank approximation.

The second set of experiments is based on a regression problem with an SVR model (9) and ε-insensitive loss, where 1,500 training samples and 1,014 test samples are generated by the Sinc function with Gaussian noise $y = \frac{\sin(x)}{x} + \zeta$ with $x \in [-4\pi, 4\pi]$ discretized by the step of 0.01 and $\zeta \sim N(0, 0.05)$. Since the ε-insensitive loss is not LS-DC loss, we compare LibSVM with UniSVM with smoothed ε-insensitive loss (20) for solving SVR (9). Here, the kernel function $\kappa(\boldsymbol{x},\boldsymbol{z}) = \exp(-\gamma\|\boldsymbol{x}-\boldsymbol{z}\|^2)$ with $\gamma = 0.5$, λ is set as 10^{-4} and $r = 50$ for UniSVM-app. The experimental results are plotted in Fig. 4.

From the experimental results in Fig. 4, we have two findings. One is that the new UniSVM can achieve better performance than LibSVM. This may be because the added

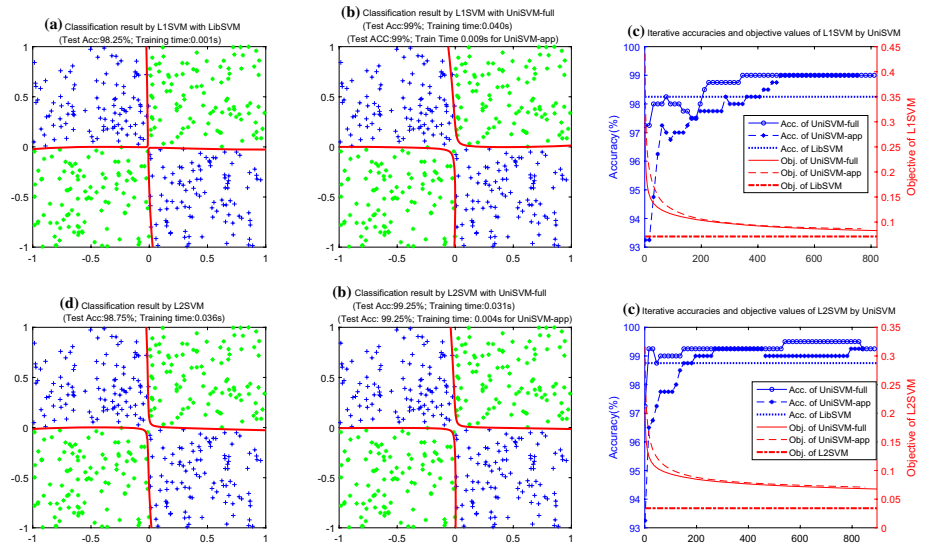


Fig. 3 Comparison of the related algorithms of SVM with convex losses. In (**a**), (**b**), (**d**) and (**e**), the classification results of the algorithms are plotted as red solid curves (since the differences between them are slight, the classification curves of UniSVM-app are not plotted and its test accuracies and the training times are correspondingly noted in (**b**) and (**e**)). In (**c**) and (**f**), the iterative processes of UniSVM are plotted. The blue curves with respect to the left y-axis are the iterative test accuracies of UniSVM, and the red curves with respect to the right y-axis are the iterative objective values of (2). The accuracies and the objective values of L1SVM/L2SVM are plotted as the horizontal lines for reference

noise follows a Gaussian distribution while UniSVM is initialized as LSSVM. The other finding is that the low-rank approximation of the kernel matrix is highly efficient here, since UniSVM-app can obtain results similar to those of UniSVM-full; this is similar to findings in Zhou (2016), Chen and Zhou (2018). The speedup rate here is less than $\frac{m}{r}$, which is because the number of iterations of UniSVM is only 3.

5.1.2 On nonconvex loss cases

The first set of experiments was again performed on the "xor" dataset as shown in Fig. 4, where some training samples (10%) are contaminated to simulate outliers by flipping labels (the test samples are noise-free). The compared algorithms include robust L1SVM in Collobert et al. (2006), MS-SVM of robust L2SVM in Tao et al. (2018), and the re-weighted L2SVM in Feng et al. (2016). The results are presented in Fig. 5, in which the classification results of L1SVM and L2SVM are listed as the references. In these experiments, only UniSVM was implemented as UniSVM-app with $r = 10$, shortened as UniSVM. The truncated parameter $a = 2$ for nonconvex loss.

From the experimental results in Fig. 5, we observe that the models based on nonconvex loss improve the classification results in cases with outliers. The algorithms in Fig. 5c–e, which need to iteratively solve L1SVM or L2SVM many times, are also affected by the outliers of the upper-right corner, where the outliers are dominated locally. However, the proposed UniSVM based on the effective LS-DC loss can completely resolve this problem. In particular, as highlighted by the results in Fig. 5d and g where the same SVM model

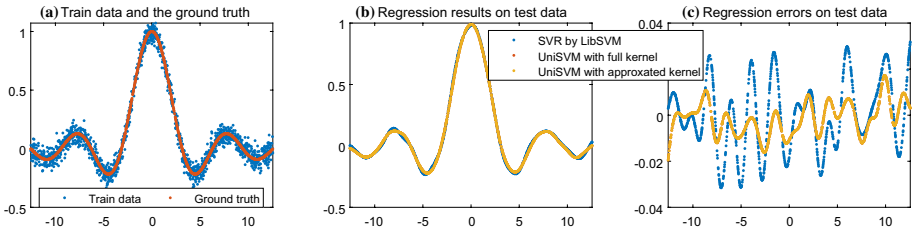


Fig. 4 The comparison of solving SVR by UniSVM with smooth ϵ-insensitive loss and LibSVM on Sinc regression problem. In (**a**), the training data and the ground truth are plotted; in (**b**), the regression results of LibSVM, UniSVM-full and UniSVM-app are given, where the respective MSRs are 0.0027, 0.0025, and 0.0025, and the training times are 0.062s, 0.144s and 0.035s; in (**c**), the regression errors of three algorithms are plotted. In (**b**) and (**c**), the difference between UniSVM-full and UniSVM-app can be neglected, while the training time of the latter is less than one-fifth of that of the former

with truncated squared hinge loss is solved by different algorithms, the proposed UniSVM can solve the robust SVM with high performance. The comparison between the results of Fig. 5e and h reveals a similar performance. The reason for this may be because the proposed UniSVM can obtain a better local minimum with a good initial point (a sparse solution of LSSVM) based on the DC decomposition of the corresponding nonconvex loss.

The second set of experiments is performed to compare the effectiveness of the related algorithms, also on the "xor" problem. The training samples are randomly generated with varied sizes from 400 to 10,000, and the test samples are generated similarly with the same sizes. The training data is contaminated by randomly flipping the labels of 10% of instances to simulate outliers. We set $r = 0.1m$ for all UniSVM algorithms to approximate the kernel matrix, and the other parameters are set as in the former experiments. The corresponding training time and test accuracies (averaged over ten trials) of the related robust SVM algorithms are plotted in Fig. 6, where the results of LibSVM (with outliers and without outliers) are also given as references.

From the experimental results in Fig. 6, we have the following findings:

- In Fig. 6a, it is clear that the performances of all the related robust SVM algorithms based on nonconvex losses are better than that of the L1SVM with convex loss on the contaminated training datasets and that all of them match the results of the noise-free case. At the same time, we notice that the differences of the test accuracies between the selected robust algorithms are very small.
- From Fig. 6b, we observe that the differences of the training time between the related algorithms are large, especially for the larger training set. The training time of the proposed UniSVM is significantly less, while the robust L1SVM (Collobert et al. 2006), Robust L2SVM (Tao et al. 2018) and re-weighted L2SVM (Feng et al. 2016)—which need to solve the constrained QP several times—have long training times. Even they can be more efficiently implemented (such as SMO) than `quadprog.m`, but their training time will be longer than that of LibSVM since all of them at least solve a QP which is similar to the QP solved by LibSVM.
- In our setting, the outliers greatly affect the results of LibSVM, not only with respect to test accuracies but also training time.
- All in all, the new proposed UniSVM with low rank kernel approximation is not only robust to noise but also highly efficient with respect to the training process.

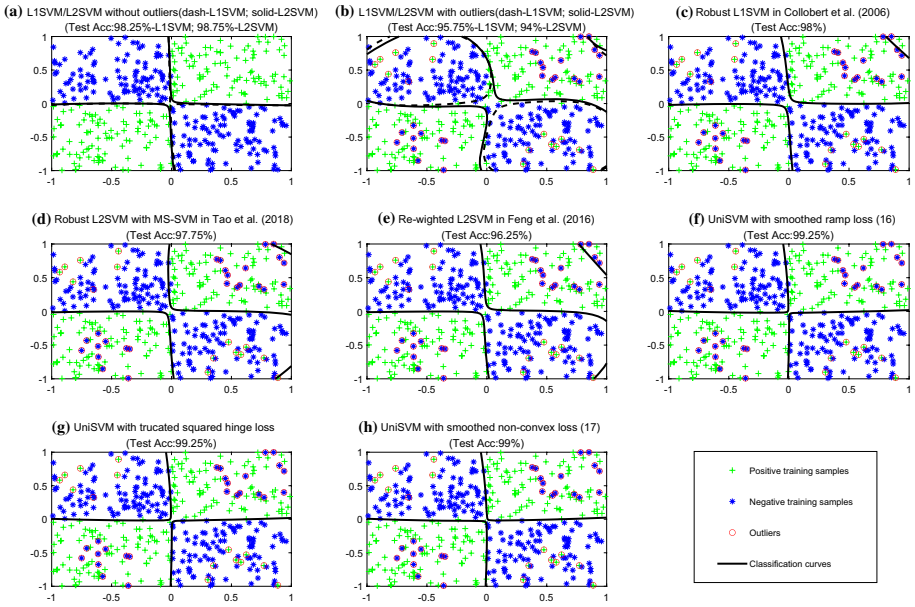


Fig. 5 Comparison of the classification results of the related algorithms of SVM with convex and non-convex losses. See the titles of the subfigures for details, where the loss (18) used in figure (**h**) with $a = b = c = 2$ is the same as (12) used in figure (**e**). It is worth noting that the results of (**d**) and (**g**) are based on the same model with the truncated squared hinge loss with the different algorithms, and (**f**) and (**i**) are based on the same model with nonconvex smooth loss (12) but are solved with a different algorithm

5.2 Experiments on larger benchmark datasets

In this section, we perform experiments to show that UniSVM can quickly train the convex and nonconvex SVM models with comparable performance using a unified scheme on large data sets. We choose only the state-of-the-art SVM tool *LibSVM* (Chang and Lin 2011) (including SVC and SVR) as the comparator, rather than other robust SVM algorithms in previous papers (Collobert et al. 2006; Wu and Liu 2007; Tao et al. 2018; Feng et al. 2016), to conserve experimental time.

First, we select four classification tasks and three regression tasks from the UCI database to illustrate the performance of the related algorithms. The detailed information of the data sets and the hyper-parameters (`training size, test size, dimension`, λ, γ) is given as follows:

Adult :	$(32561, 16281, 123, 10^{-5}, 2^{-10})$,	**Ijcnn** :	$(49990, 91790, 22, 10^{-5}, 2^{0})$,
Shuttle :	$(43500, 14500, 9, 10^{-5}, 2^{1})$,	**Vechile** :	$(78823, 19705, 100, 10^{-2}, 2^{-3})$;
Cadata :	$(10640, 10000, 8, 10^{-2}, 2^{0})$,	**3D** – *Spatial* :	$(234874, 200000, 3, 10^{-3}, 2^{6})$,
Slice :	$(43500, 10000, 385, 10^{-9}, 2^{-5})$.		

Here, the classification tasks have the default splitting, and the regression tasks are split randomly. The λ (regularizer) and γ (for Gaussian kernel $\kappa(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\gamma\|\boldsymbol{x} - \boldsymbol{z}\|^2)$) are roughly chosen by the grid search. For the parameters of loss functions, we simply use the default value (given next). Fine-tuning all parameters will certainly improve the performance further.

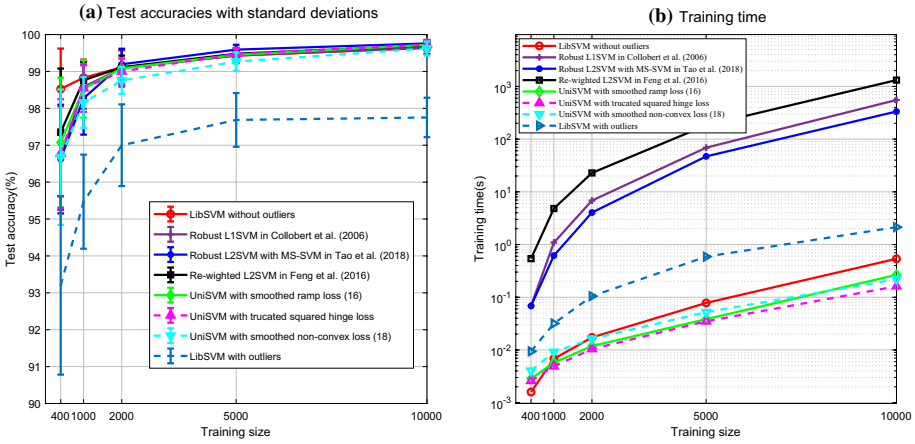


Fig. 6 The comparisons of the training time and test accuracies (averaged over ten trials) of the related algorithms on the contaminated "xor" datasets with different training sizes

To implement the UniSVM for larger training data, we use the pivoted Cholesky factorization method proposed in Zhou (2016) to approximate the kernel matrix $\boldsymbol{K}$, and the low-rank approximation error is controlled by the first matched criterion $trace(\boldsymbol{K} - \tilde{\boldsymbol{K}}) < 0.001 \cdot m$ or $r \leq 1000$, where m is the training size and r is the upper bound of the rank.

The first set of experiments shows that the proposed UniSVM can train SVM models with convex or nonconvex loss for classification problems. The chosen losses for UniSVM_1 to UniSVM_{10} are listed as following:

UniSVM_1 : least squares loss,
UniSVM_2 : smoothed hinge ($p = 10$),
UniSVM_3 : squared hinge loss,
UniSVM_4 : truncated squared hinge ($a = 2$),
UniSVM_5 : truncated least squares ($a = 2$),
UniSVM_6 : loss (16)($a = 2$),
UniSVM_7 : loss (17)($p = 10$),
UniSVM_8 : loss (18)($a = b = c = 2$),
UniSVM_9 : loss (18)($a = b = 2, c = 4$),
UniSVM_{10} : loss (18s)($a = 2, b = 3, c = 4$).

The results in Table 1 are obtained based on the original data sets, and those of Table 2 are based on the contaminated data sets, where 20% of the labels of training instances are randomly flipped. Since the kernel approximation in Zhou (2016) undergoes random initialization, the training times recorded in Matlab are not very stable, and since random flipping is observed for noise cases, all results are averaged over ten random trials.

From the results in Tables 1 and 2, we draw the following conclusions:

- UniSVMs with different losses work well using a unified scheme in all cases. They are mostly faster than LibSVM and offer comparable performance in noise-free cases. The training time of LibSVM in Table 2 is notably longer than its training time in Table 1 because the flipping process adds a large number of support vectors. However, owing to the sparse solution of (28), this influence on UniSVMs is quite weak.
- Comparing the training time (including the time to obtain $\boldsymbol{P}$ for approximating the kernel matrix $\boldsymbol{K}$) of UniSVM_1 (least squares) with other methods, it is clear that the proposed UniSVM requires a very low cost after the first iteration, as other UniSVMs always run UniSVM_1 in their first iteration.

- All the UniSVMs with nonconvex losses are working as efficiently as those with convex losses. Particularly, the UniSVMs with nonconvex losses maintain high performance on the contaminated data sets. The new proposed loss (18) with two more parameters always achieves the highest performance.

The second set of experiments examined the performance of the UniSVM for solving regression tasks with convex and nonconvex losses. The experimental results are listed in Table 3. The chosen losses for UniSVM_1 to UniSVM_6 are listed as follows:

UniSVM_1 : least squares loss, UniSVM_2 : smoothed $\varepsilon-$ insensitive loss (20)($p = 100$),
UniSVM_3 : Huber loss ($\delta = 0.1$), UniSVM_4 : smoothed absolute loss ($p = 100$),
UniSVM_5 : truncated least squares ($a = 2$), UniSVM_6 : truncated Huber loss ($\delta = 0.1, a = 2$).

From the results in Table 3, it is again observed that UniSVMs with different losses work well for a unified scheme. All of them are more efficient than LibSVM with comparable performance. For example, LibSVM requires a very long training time on the second 3D-Spatial task because of excessive training samples, and LibSVM cannot finish the task on the last Slice data set, possibly because of an excessive number of support vectors. In two cases, all UniSVMs function well and exhibit comparable performance, which is primarily attributed to the efficient low-rank approximation of the kernel matrix. It is also noted that the UniSVMs with nonconvex losses function as efficiently as those with convex loss.

In the third set of experiments, we challenge UniSVM with two classification tasks on very large data sets (up to millions of samples) on the same computer. The selected data sets are:

- `Covtype`: a binary class problem with 581,012 samples, where each example has 54 features. We randomly split it into 381,012 training samples and 200,000 test samples. The parameters used are $\gamma = 2^{-2}$ and $\lambda = 10^{-8}$.
- `Checkerboard3M`: based on the noise-free version of the 2-dimensional Checkerboard data set (4×4-grid XOR problem), which was widely used to show the effectiveness of nonlinear kernel methods. The dataset was sampled by uniformly discretizing the regions $[0, 1] \times [0, 1]$ to $2000^2 = 4000000$ points and labeling two classes by the 4×4-grid XOR problem, and was then split randomly into 3,000,000 training samples and 1,000,000 test samples. The parameters used are $\gamma = 2^4$ and $\lambda = 10^{-7}$.

Those datasets are also used in Zhou (2016). Because of the limited memory of our computer, the kernel matrix on `Covtype` is approximated as $\boldsymbol{P}\boldsymbol{P}^\top$ with $\boldsymbol{P} \in \mathbb{R}^{m\times 1000}$, and the kernel matrix on `Checkerboard3M` is approximated as $\boldsymbol{P}\boldsymbol{P}^\top$ with $\boldsymbol{P} \in \mathbb{R}^{m\times 300}$, where m is the training size. The experimental results are given in Table 4, where LibSVM cannot accomplish the tasks because of its long training time. The losses used in the algorithms are the same as those in Table 1.

From the results in Table 4, we observe that the UniSVM works well on very large data sets. We also reach conclusions which are consistent with the results in Tables 1 and 2 . For example, UniSVMs with different losses work well using a unified scheme and offer comparable performance, and all of the UniSVMs with nonconvex losses function as efficiently as those with convex losses. Particularly, the UniSVMs with nonconvex losses

Table 1 *Classification tasks I*–Test accuracies and the training times of the related algorithms on the benchmark data sets. All results are averaged over ten trials with the standard deviations in brackets; the first four lines are based on convex losses and the others are based on nonconvex losses

	Test accuracy (%)				Training time (CPU seconds)			
Algorithm	Adult	Ijcnn	Shuttle	Vechile	Adult	Ijcnn	Shuttle	Vechile
LibSVM	84.65(0.00)	98.40(0.00)	99.81(0.00)	84.40(0.00)	51.49(0.07)	23.02(0.63)	4.75(0.16)	1091(175)
UniSVM_1	84.56(0.02)	94.65(0.07)	98.80(0.04)	85.24(0.01)	**0.44**(0.02)	**18.24**(0.15)	**0.34**(0.02)	**34.77**(0.44)
UniSVM_2	84.68(0.04)	97.07(0.05)	99.81(0.01)	84.42(0.00)	0.98(0.04)	38.51(0.32)	2.44(0.10)	36.66(0.62)
UniSVM_3	85.13(0.02)	98.22(0.03)	99.82(0.00)	85.23(0.01)	0.62(0.03)	35.40(0.28)	2.03(0.06)	35.34(0.39)
UniSVM_4	84.75(0.04)	98.25(0.04)	99.82(0.00)	84.72(0.00)	1.13(0.04)	38.56(0.53)	2.47(0.06)	37.20(0.42)
UniSVM_5	83.32(0.05)	94.59(0.08)	98.81(0.04)	84.71(0.00)	0.58(0.03)	19.18(0.35)	0.38(0.02)	36.81(0.52)
UniSVM_6	84.82(0.02)	98.20(0.03)	99.82(0.00)	84.70(0.00)	1.08(0.03)	40.11(0.44)	2.72(0.10)	36.65(0.34)
UniSVM_7	84.20(0.02)	97.13(0.05)	99.82(0.00)	84.43(0.00)	1.38(0.04)	40.40(0.51)	2.71(0.06)	37.61(0.35)
UniSVM_8	84.75(0.02)	97.84(0.04)	99.82(0.00)	84.53(0.00)	0.97(0.04)	38.13(0.26)	2.40(0.11)	36.10(0.45)
UniSVM_9	85.09(0.02)	**98.46**(0.04)	**99.83**(0.00)	85.34(0.00)	1.15(0.05)	36.95(0.29)	3.30(0.11)	36.69(0.44)
UniSVM_{10}	**85.16**(0.03)	98.36(0.03)	99.82(0.00)	**85.49**(0.00)	0.84(0.03)	33.92(0.22)	2.90(0.15)	36.47(0.55)

Table 2 *Classification tasks II*–Test accuracies and the training times of the related algorithms on the benchmark data sets with *flipping 20% of training data labels*. All results are averaged over ten trials with the standard deviations in brackets; the first four lines are based on convex losses and the others are based on nonconvex losses

	Test accuracy (%)				Training time (CPU seconds)			
Algorithm	Adult	Ijcnn	Shuttle	Vechile	Adult	Ijcnn	Shuttle	Vechile
LibSVM	78.25(0.00)	93.80(0.00)	98.89(0.00)	84.28(0.04)	104.0(0.8)	191.7(1.2)	82.28(1.52)	1772(123)
UniSVM1	84.55(0.02)	93.90(0.08)	98.71(0.04)	85.19(0.00)	**0.44**(0.06)	**18.34**(0.33)	**0.35**(0.02)	**34.76**(0.43)
UniSVM2	82.27(0.06)	93.72(0.04)	99.01(0.10)	84.25(0.00)	0.65(0.07)	20.56(0.39)	0.60(0.04)	36.99(0.43)
UniSVM3	84.55(0.02)	93.95(0.08)	98.72(0.04)	85.19(0.00)	0.45(0.06)	18.74(0.34)	0.38(0.02)	34.94(0.44)
UniSVM4	84.26(0.03)	97.36(0.05)	99.81(0.00)	84.37(0.00)	1.23(0.09)	40.23(2.85)	2.52(0.09)	37.53(0.46)
UniSVM5	82.62(0.03)	93.96(0.03)	98.81(0.02)	84.34(0.00)	0.60(0.05)	20.25(0.48)	0.39(0.02)	37.50(0.48)
UniSVM6	84.25(0.02)	97.59(0.04)	99.81(0.00)	84.46(0.00)	1.14(0.08)	48.74(2.55)	2.72(0.14)	37.19(0.42)
UniSVM7	83.80(0.04)	96.05(0.04)	99.67(0.06)	84.28(0.00)	1.52(0.08)	46.14(1.77)	2.67(0.11)	38.14(0.44)
UniSVM8	84.38(0.04)	94.76(0.05)	99.25(0.10)	84.39(0.00)	0.76(0.07)	23.32(0.48)	0.99(0.05)	36.53(0.44)
UniSVM9	**85.09**(0.02)	**97.68**(0.04)	**99.80**(0.00)	85.31(0.00)	1.06(0.07)	44.67(2.99)	4.29(0.22)	36.55(0.48)
UniSVM10	84.83(0.02)	95.77(0.09)	99.44(0.05)	**85.46**(0.00)	0.56(0.06)	21.26(0.47)	0.81(0.04)	35.69(0.46)

maintain high performance because many contaminated samples may exist in very large training cases.

6 Conclusion and future work

In this work, we first define a kind of LS-DC loss with an effective DC decomposition. Based on the DCA procedure, we then propose a unified algorithm (UniSVM) for training SVM models with different losses for both classification problems and regression

Table 3 *Regression task*–Test RMSE (root-mean-square-error) and the training time of the related algorithms on the benchmark data sets. All results are averaged over ten trials with the standard deviations in brackets; The first four lines are based on convex losses and the rest are based on the truncated nonconvex losses

	Test RMSE			Training time (CPU seconds)		
Algorithm	Cadata	3D-Spatial	Slice	Cadata	3D-Spatial	Slice
LibSVM	0.314(0.000)	0.464(0.000)	—	3.38(0.04)	4165(2166)	$> 3hr$
UniSVM_1	0.314(0.000)	0.455(0.000)	**6.725**(0.101)	1.06(0.06)	**96.3**(5.0)	**25.40**(0.08)
UniSVM_2	**0.307**(0.000)	0.459(0.000)	6.753(0.100)	1.38(0.07)	118.9(4.2)	44.75(1.05)
UniSVM_3	0.310(0.000)	0.463(0.000)	6.870(0.103)	1.31(0.09)	112.8(3.8)	60.82(1.70)
UniSVM_4	0.308(0.000)	0.464(0.000)	6.765(0.100)	1.44(0.08)	123.3(4.3)	50.89(1.25)
UniSVM_5	0.315(0.000)	**0.454**(0.000)	6.868(0.116)	1.10(0.06)	99.6(4.1)	83.75(13.30)
UniSVM_6	0.312(0.000)	0.465(0.000)	6.775(0.105)	1.32(0.07)	121.5(4.9)	75.72(4.45)

problems. Particularly, for training robust SVM models with nonconvex losses, UniSVM has a dominant advantage over all the existing algorithms because it always has a closed-form solution per iteration, while the existing ones must solve a constraint programming per iteration. Furthermore, UniSVM can solve large-scale nonlinear problems efficiently after the kernel matrix has the low-rank matrix approximation.

Several experimental results verify the efficacy and feasibility of the proposed algorithm. The most prominent advantage of the proposed algorithm is that it can be easily grasped by users or researchers since its core code in Matlab is less than 10 lines (See Appendix C).

In this work, we mainly discussed the methods to deal with the (convex or nonconvex) loss of the regularized loss minimization (Shalev-Shwartz and Ben-David 2014) by DCA to enhance the sparseness of the samples or robustness of the learner. However, there are also some works which handle the nonconvex regularizer part of the regularized loss minimization by DCA, which can strengthen the sparseness of the features and serve as a highly efficient tool for feature selection. For example, in Neumann et al. (2004), Le Thi et al. (2008, 2009), Ong and Le Thi (2013), some smooth approximations of the nonconvex "ℓ_0 norm" are decomposed as DC forms, then DCA is used to perform feature selection and many satisfactory results are produced. We will intensively study whether or not our new LS-DC decomposition can improve those kinds of learning problems.

Appendix

Appendix A: The proof of propositions

The proof of propositions 1

Proof We illustrate them one by one.

Table 4 *Classification III*–Test accuracies and training times of the related algorithms on two very large data sets, `Covtype` and `Checkerboard3M`, where all results are averaged over *five* with the standard deviations in brackets. The first three lines are based on convex losses and the others are based on nonconvex losses

	Test accuracy (%)		Training time (CPU seconds)	
Algorithm	Covtype	Checkerboard3M	Covtype	Checkerboard3M
UniSVM_1	81.11(0.02)	98.04(0.08)	**183.68**(11.80)	**37.94**(2.68)
UniSVM_2	80.80(0.03)	98.05(0.18)	205.92(12.77)	77.28(2.73)
UniSVM_3	81.14(0.02)	98.07(0.08)	188.40(12.17)	40.72(2.67)
UniSVM_4	83.15(0.12)	99.94(0.01)	540.00(84.54)	634.54(45.18)
UniSVM_5	81.46(0.04)	97.99(0.07)	224.34(15.00)	42.53(2.87)
UniSVM_6	83.25(0.14)	99.94(0.01)	449.73(42.14)	574.43(4.22)
UniSVM_7	82.90(0.10)	99.83(0.03)	405.50(11.54)	545.35(3.81)
UniSVM_8	82.19(0.09)	99.90(0.01)	282.55(16.65)	580.34(3.76)
UniSVM_9	**83.40**(0.05)	**99.95**(0.01)	409.71(44.09)	693.48(4.30)
UniSVM_{10}	81.89(0.03)	99.94(0.02)	269.06(10.15)	777.14(8.63)

(a) It is clear.

(b) It is because $u^2 - \min\{u^2, a\} = (u^2 - a)_+$ is a convex function.

(c) It is because $Au^2 - u_+^2$ with $A \geq 1$ is a convex function.

(d) It is because $Au^2 - \min\{u_+^2, a\} = Au^2 - u_+^2 + (u_+^2 - a)_+$ with $A \geq 1$ is convex.

(e) First we show that the hinge loss $\ell(y,t) = (1-yt)_+$ is not an LS-DC loss. If let $g(u) = Au^2 - u_+$, we have $g'_-(0) = 0 > g'_+(0) = -1$. Hence by Theorem 24.1 in Rockafellar (1972), we conclude that $g(u)$ is not convex for all A $(0 < A < +\infty)$.

Notice that $(1-yt)_+ = \lim_{p\to+\infty} \frac{1}{p}\log(1+\exp(p(1-yt)))$. Let $\psi(u) = \frac{1}{p}\log(1+\exp(pu))$, and we have $\psi''(u) = \frac{p\exp(pu)}{(1+\exp(pu))^2} \leq \frac{p}{4}$. By Theorem 1, we know that $\ell_p(y,t) = \frac{1}{p}\log(1+\exp(p(1-yt)))$ is an LS-DC loss with $A \geq p/8$. In experiments, letting $1 \leq p \leq 100$, $\ell_p(y,t)$ is a good approximation of the hinge loss.

(f) The reason that the ramp loss is **not** an LS-DC loss is the same as that of the hinge loss. Its two smoothed approximations (16) and (17) are LS-DC loss. The proof of the first is similar to that of the squared hinge loss and the proof of the second is similar to that of the approximation of the hinge loss.

(g) Let $g(u) = Au^2 - a\Big(1 - \exp(-\frac{1}{a}u_+^2)\Big)$. Then $g(u)$ is a convex function because $g'(u) = 2Au - 2u_+\exp(-\frac{1}{a}u_+^2))$ is monotonically increasing if $A \geq 1$.

(h) Let $\psi(u) = a(1 - e^{-\frac{1}{b}u_+^c})$. If $c = 2$, set $A \geq \frac{a}{b} = \frac{1}{2}M(a,b,2)$ and let $g(u) = Au^2 - \psi(u)$. Hence $g(u)$ is convex because $g'(u) = 2Au - \frac{2a}{b}u_+e^{-\frac{1}{b}u_+^2}$ is monotonically increasing.

For $c > 2$, $\psi(u)$ is second-order derivable, and according to Theorem 1 we only need to obtain the upper bound of $\psi''(u)$. If $u \leq 0$, we have $\psi''(u) = 0$. If $u > 0$, then $\psi''(u) = \frac{ac}{b}\Big((c-1)u^{c-2} - \frac{c}{b}u^{2c-2}\Big)e^{-\frac{1}{b}u^c}$ and

$$\psi'''(u) = \frac{ac}{b}u^{c-3}e^{-\frac{1}{b}u^c}\left(\frac{c^2}{b^2}u^{2c} - \frac{3c(c-1)}{b}u^c + (c-1)(c-2)\right).$$

Letting $\psi'''(u) = 0$, we get the roots u_1^* and u_2^* $(0 < u_1^* < u_2^*)$, where

$$u_1^* = (b \cdot h(c))^{\frac{1}{c}}$$

with $h(c) = (3(c-1) - \sqrt{5c^2 - 6c + 1})/(2c)$, which is a local maximum of $\psi''(u)$. Noting that $\lim_{u\to 0} \psi''(u) = \lim_{u\to\infty} \psi''(u) = 0$, we have that the global maximum of $\psi''(u)$ reaches at u_1^*. Putting u_1^* in $\psi''(u)$, we prove that $\psi''(u) \le \psi''(u_1^*) =: M(a,b,c)$ for any u, where $M(a,b,c) = \frac{ac}{b^{2/c}}\big((c-1)(h(c))^{1-2/c} - c(h(c))^{2-2/c}\big)e^{-h(c)}$.

For example, $M(2,2,2) = 2, M(2,2,4) \approx 4.5707 < 5, M(2,3,4) \approx 3.7319 < 4$. Thus, the parameter $A = \frac{1}{2}M(a,b,c)$ is not very large.

□

The proof of propositions 2

Proof We illustrate them one by one.

(1) It is clear.

(2) The ε-insensitive loss $\ell_\varepsilon(y,t) := (|y-t| - \varepsilon)_+$ is not an LS-DC loss. The reason is similar to that of the hinge loss in item (e). However, its smoothed approximation (20) is an LS-DC loss with $A \ge p/4$. Let $\psi(u) = \frac{1}{p}\log(1+\exp(-p(u+\varepsilon))) + \frac{1}{p}\log(1+\exp(p(u-\varepsilon)))$. We have

$$\psi''(u) = \frac{p\exp(-p(u+\varepsilon))}{(1+\exp(-p(u+\varepsilon)))^2} + \frac{p\exp(p(u-\varepsilon))}{(1+\exp(p(u-\varepsilon)))^2} \le \frac{p}{2}.$$

(3) The absolute loss $\ell(y,t) = |y-t|$ is also not an LS-DC loss. Clearly, the Huber loss $\ell_\delta(y,t)$ which approximates the absolute loss, is an LS-DC loss with $A \ge 1/(2\delta)$; setting $\varepsilon = 0$ in (20) we obtain another smoothed absolute loss, which is an LS-DC loss with $A \ge p/4$.

(4) It is clear.

□

Appendix B: The lists of some related losses and their subdifferentials

The most related losses and their subdifferentials for updating $\boldsymbol{\gamma}^k$ by (23) are listed in Table 5. The LS-DC parameters of the LS-DC losses are also given in the last column. In experiments, we always use the lower-bound of the parameter.

Table 5 The list of the losses and their subdifferentials

Loss name	Classification losses: $\ell(y,t)=\psi(1-yt)$ and $\frac{\partial}{\partial t}\ell(y,t)=-y\partial\psi(1-yt)$.		
	$\psi(u)$	$\partial\psi(u)$	A
Least squares loss	$\psi^{(1)}(u) := u^2$	$\partial\psi(u) := 2u$	≥ 1
Truncated Least squares loss	$\psi_a^{(2)}(u) := \min\{u^2, a\}$	$\partial\psi_a(u) := \begin{cases} 2u, & \lvert u\rvert < \sqrt{a}, \\ 0, & \lvert u\rvert \geq \sqrt{a}, \end{cases}$	
Squared hinge loss	$\psi^{(3)}(u) := u_+^2$	$\partial\psi(u) := 2u_+$	≥ 1
Truncated squared hinge loss	$\psi_a^{(4)}(u) := \min\{u_+^2, a\}$	$\partial\psi_a(u) := \begin{cases} 2u, & 0 < u < \sqrt{a}, \\ 0, & others, \end{cases}$	
Hinge loss	$\psi^{(5)}(u) := u_+$, NOT LS-DC loss, smoothed by $\psi^{(6)}$.		
Smooth Hinge loss	$\psi_p^{(6)}(u) := u_+ + \frac{\log\left(1+e^{-p\lvert u\rvert}\right)}{p}$	$\partial\psi_p(u) := \frac{\min\{1, e^{pu}\}}{(1+e^{-p\lvert u\rvert})}$	$\geq \frac{p}{8}$
Ramp loss	$\psi_a^{(7)}(u) := \min\{u_+, a\}$, NOT LS-DC loss, smoothed by $\psi^{(8)}$ and $\psi^{(9)}$.		
Smoothed ramp loss 1	$\psi_a^{(8)}(u) := \begin{cases} \frac{2}{a}u_+^2, & u \leq \frac{a}{2}, \\ a - \frac{2}{a}(a-u)_+^2, & u > \frac{a}{2}, \end{cases}$	$\partial\psi_a(u) := \begin{cases} \frac{4}{a}u_+, & u \leq \frac{a}{2}, \\ \frac{4}{a}(a-u)_+, & u > \frac{a}{2}, \end{cases}$	
Smoothed ramp loss 2	$\psi_{(a,p)}^{(9)}(u) := \frac{1}{p}\log\left(\frac{1+e^{pu}}{1+e^{p(u-a)}}\right)$	$\partial\psi_{(a,p)}(u) := \frac{e^{-p(u-a)}-e^{-pu}}{(1+e^{-p(u-a)})(1+e^{-pu})}$	$\geq \frac{p}{8}$
Smoothed nonconvex loss(18)	$\psi_{(a,b,c)}^{(10)}(u) := a\left(1-e^{-\frac{1}{b}u_+^c}\right)$	$\partial\psi_{(a,b,c)}(u) := \frac{ac}{b}u_+^{c-1}e^{-\frac{1}{b}u_+^c}$	$\geq \frac{1}{2}M(a,b,c)$ See (19)
	Regression losses: $\ell(y,t)=\tilde{\psi}(y-t)$ and $\frac{\partial}{\partial t}\ell(y,t)=-\partial\tilde{\psi}(y-t)$.		
Least squares loss	$\tilde{\psi}^{(1)}(u) := u^2$	$\partial\tilde{\psi}(u) := 2u$	≥ 1
Truncated least squares loss	$\tilde{\psi}_a^{(2)}(u) := \min\{u^2, a\}$	$\partial\tilde{\psi}_a(u) := \begin{cases} 2u, & \lvert u\rvert < \sqrt{a}, \\ 0, & \lvert u\rvert \geq \sqrt{a}, \end{cases}$	≥ 1
ε-insensitive loss	$\tilde{\psi}_\varepsilon^{(3)}(u) := (\lvert u\rvert - \varepsilon)_+$, NOT LS-DC loss, smoothed by $\tilde{\psi}^{(4)}$.		
Smoothed ε-insensitive loss	$\tilde{\psi}_{(p,\varepsilon)}^{(4)}(u) := \frac{\log((1+e^{-p(u+\varepsilon)})(1+e^{p(u-\varepsilon)}))}{p}$	$\partial\tilde{\psi}_{(p,\varepsilon)}(u) := \frac{1}{1+e^{p(u-\varepsilon)}} - \frac{1}{1+e^{-p(\varepsilon+u)}}$	$\geq \frac{p}{4}$
Absolute loss	$\tilde{\psi}^{(5)}(u) := \lvert u\rvert$, NOT LS-DC loss, smoothed by $\tilde{\psi}^{(6)}$ and $\tilde{\psi}^{(7)}$.		

Table 5 (continued)

Loss name	Classification losses: $\ell(y,t)=\psi(1-yt)$ and $\frac{\partial}{\partial t}\ell(y,t)=-y\partial\psi(1-yt)$.		
	$\psi(u)$	$\partial\psi(u)$	A
Huber loss	$\tilde{\psi}^{(6)}_{\delta}(u) := \begin{cases} \frac{1}{2\delta}u^2, & \lvert u\rvert < \delta, \\ \lvert u\rvert - \frac{\delta}{2}, & \lvert u\rvert \geq \delta, \end{cases}$	$\partial\tilde{\psi}_{\delta}(u) := \begin{cases} \frac{1}{2\delta}u, & \lvert u\rvert < \delta, \\ \operatorname{sgn}(u), & \lvert u\rvert \geq \delta, \end{cases}$	$\geq \frac{1}{2\delta}$
Smoothed Absolute loss	$\tilde{\psi}^{(7)}_{p}(u) := \frac{1}{p}\log((1+e^{-pu})(1+e^{pu}))$	$\partial\tilde{\psi}_{p}(u) := \frac{\min\{1,e^{pu}\}-\min\{1,e^{-pu}\}}{1+e^{-p\lvert u\rvert}}$	$\geq \frac{p}{4}$
Truncated Hubber loss	$\tilde{\psi}^{(8)}_{(\delta,a)}(u) := \min\{\tilde{\psi}^{(6)}_{\delta}(u), a\}$	$\partial\tilde{\psi}_{\delta}(u) := \begin{cases} \frac{1}{2\delta}u, & \lvert u\rvert < \delta, \\ \operatorname{sgn}(u), & \delta \leq \lvert u\rvert \leq a, \\ 0, & \lvert u\rvert > a. \end{cases}$	$\geq \frac{1}{2\delta}$

Appendix C: Matlab code for UniSVM

Matlab code for solving UniSVM with the full kernel matrix available in Subsect. 4.1 is listed as follows; please see the notes for other cases. The demo codes can also be found at https://github.com/stayones/code-UNiSVM.

```
0: function [alpha] = UniSVM_small(K, y, lambda, A, dloss, eps0)
1: %K-kernel matrix; y-targets; lambda-regularizer;
   %A-parameter of the LS-DC loss; dloss-the derivative function of the LS-DC loss.
2: m = length(y);  v_old = zeros(m,1);
3: Q = inv(K + lambda * m / A * eye(m));alpha = Q*y; %This is the LS-SVM solution.
4: while 1
5:    Ka =  K * alpha;
6:    v= - y .* dloss(1-y .* Ka); %for CLASSIFICATION task;
      % v= - dloss(y - Ka);       %for REGRESSION task;
7:    if norm(v_old - v) < eps0, break; end
8:    alpha = Q * (Ka - v *(0.5/A)) ;  v_old = v;%
9: end
   return

Note:
1) With squared hinge loss, dloss(u)=2*max(u,0);
   With truncated squared hinge loss, dloss(u)=2*max(u,0).*(u<=sqrt(a));
   With truncated least squares loss, dloss(u)=2*(u).*(abs(u)<=sqrt(a));
   With other losses, dloss(u) given in the Table 5 in Appendix B.
2) For large training problem, the input K is taken place as P and B with K=P*P',
   then make the following revisions:
   Line 3 --> Q=inv((lambda*m/A*eye(length(B)) + P'*P)*P(B,:)');alpha = Q*(P'*y);
   Line 5 --> Ka = P*(P(B,:)'*alpha);
   Line 8 --> alpha = Q*(P'*(Ka - v *(0.5/A))); v_old = v.
```

Acknowledgements We would like to acknowledge support for this project from the National Natural Science Foundation of China under Grant No. 61772020. We also thank the anonymous reviewers for their useful comments that greatly improved the presentation.

Declarations

conflict of interest The authors declare that they have no conflict of interest.

References

Boyd, S. P., & Vandenberghe, L. (2009). *Convex optimization* (7th ed.). Cambridge University Press.

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

Chen, L., & Zhou, S. (2018). Sparse algorithm for robust lssvm in primal space. *Neurocomputing, 257*(31), 2880–2891. https://doi.org/10.1016/j.neucom.2017.10.011.

Chen, P. H., Fan, R. E., & Lin, C. J. (2006). A study on SMO-type decomposition methods for support vector machines. *IEEE Transactions on Neural Networks, 17*(4), 893–908.

Collobert, R., Sinz, F., Weston, J., & Bottou, L. (2006). Trading convexity for scalability. In: Proceedings of the 23rd international conference on Machine learning - ICML'06, ACM Press, pp 201–208.

Feng, Y., Yang, Y., & Huang, X. (2016). Robust support vector machines for classification with nonconvex and smooth losses. *Neural Computation, 28*(6), 1217–1247.

Golub, G. H., & Loan, C. F. V. (1996). *Matrix Computations*. The John Hopkins University Press.

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., & Stahel, W. A. (2011). *Robuststatistics: The approach based on influence functions*. Wiley.

Jiao, L., Bo, L., & Wang, L. (2007). Fast sparse approximation for least squares support vector machines. *IEEE Transactions on Neural Networks, 18*(3), 685–697.

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

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

Le Thi, H. A., Le Hoai, M., Nguyen, V. V., & Pham Dinh, T. (2008). A DC programming approach for feature selection in support vector machines learning. *Journal of Advances in Data Analysis and Classification, 2*(3), 259–278.

Le Thi, H. A., Nguyen, V. V., & Ouchani, S. (2009). Gene selection for cancer classification using DCA. *Journal of Frontiers of Computer Science and Technology, 3*(6), 612–620.

Le Thi, H. A., & Pham, Dinh T. (2018). DC programming and DCA: Thirty years of developments. *Mathematical Programming, 169*(1), 5–68.

Neumann, J., Schnrr, C., & Steidl, G. (2004). SVM-based feature selection by direct objective minimization. In: *Lecture Notes in Computer Science* (Vol. 3175, pp. 212–219) Springer.

Naderi, S., He, K., Aghajani, R., Sclaroff, S., & Felzenszwalb, P. (2019). Generalized majorization-minimization. In: Proceedings of the 36-th International Conference on Machine Learning, Long Beach, CA, USA.

Ong, C. S., & Le Thi, H. A. (2013). Learning sparse classifiers with difference of convex functions algorithms. *Optimization Methods and Software, 28*(4), 830–854.

Ong, C.S., Mary, X., Canu, S., & Smola, A.J. (2004). Learning with non-positive kernels. In: Twenty-first International Conference on Machine learning - ICML'04, ACM Press.

Pham Dinh, T., & El-Bernoussi, S. (1986). Algorithms for solving a class of nonconvex optimization problems. methods of subgradients. In: Fermat Days 85: Mathematics for Optimization, Elsevier, pp 249–271.

Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. In C. J. Burges & A. J. Smola (Eds.), *Advances in Kernel method-support vector learning* (pp. 185–208). MIT Press.

Rockafellar, R. T. (1972). *Convex analysis* (2nd ed.) Princeton University Press.

Schölkopf, B., & Smola, A. J. (2002). *Learning with Kernels-support vector machines, regularization, optimization and beyond*. The MIT Press.

Schölkopf, B., Herbrich, R., & Smola, A.J. (2001). A generalized representer theorem. In: Proceedings of the Annual Conference on Computational Learning Theory, Springer, pp 416–426.

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

Shen, X., Tseng, G. C., Zhang, X., & Wong, W. H. (2003). On ϕ-learning. *Journal of the American Statistical Association, 98*(463), 724–734.

Steinwart, I. (2003). Sparseness of support vector machines. *Journal of Machine Learning Research, 4,* 1071–1105.

Steinwart, I., & Christmann, A. (2008). *Support vector machines*. Springer-Verlag.

Steinwart, I., Hush, D., & Scovel, C. (2011). Training SVMs without offset. *Journal of Machine Learning Research, 12,* 141–202.

Sun, S., Zhao, J., & Zhu, J. (2015). A review of nyström methods for large-scale machine learning. *Information Fusion, 26,* 36–48.

Suykens, J. A. K., & Vandewalle, J. (1999a). Least square spport vector machine classifiers. *Neural Processing Letters, 9*(3), 293–300.

Suykens, J. A. K., & Vandewalle, J. (1999b). Least squares support vector machine classifiers. *Neural Processing Letters, 9*(3), 293–300.

Suykens, J. A. K., Gestel, T. V., Brabanter, J. D., Moor, B. D., & Vandewalle, J. (2002). *Least squares Support Vector Machines*. World Scientific.

Tao, Q., Wu, G., & Chu, D. (2018). Improving sparsity and scalability in regularized nonconvex truncated-loss learning problems. *IEEE Transactions on Neural Networks and Learning Systems, 29*(7), 2782–2793.

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE Transactions on Neural Network, 10*(5), 988–999.

Vapnik, V. N. (2000). *The nature of dtatistical learning theory*. Springer-Verlag.

Wu, Y., & Liu, Y. (2007). Robust truncated hinge loss support vector machines. *Publications of the American Statistical Association, 102*(479), 974–983.

Xu, H.M., Xue, H., Chen, X.H., & Wang, Y.Y. (2017). Solving indefinite kernel support vector machine with difference of convex functions programming. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI Press, AAAI'17, pp 2782–2788.

Yuille, A. L., & Rangarajan, A. (2003). The concave-convex procedure. *Neural Computation, 15*(4), 915–936.

Zhou, S. (2013). Which is better? regularization in RKHS vs $\mathfrak{R}^m$ on reduced SVMs. *Statistics, Optimization and Information Computing, 1*(1), 82–106.

Zhou, S. (2016). Sparse LSSVM in primal using Cholesky factorization for large-scale problems. *IEEE Transactions on Neural Networks and Learning Systems, 27*(4), 783–795.

Zhou, S., Liu, H., Ye, F., & Zhou, L. (2009). A new iterative algorithm training SVM. *Optimization Methods and Software, 24*(6), 913–932.

Zhou, S., Cui, J., Ye, F., Liu, H., & Zhu, Q. (2013). New smoothing SVM algorithm with tight error bound and efficient reduced techniques. *Computational Optimization and Applications, 56*(3), 599–618.