International Journal of Machine Learning and Cybernetics (2019) 10:1173–1187
https://doi.org/10.1007/s13042-018-0796-7

ORIGINAL ARTICLE

A fast iterative algorithm for support vector data description

Songfeng Zheng[1]

Received: 9 February 2017 / Accepted: 26 February 2018 / Published online: 5 March 2018

Abstract
Support vector data description (SVDD) is a well known model for pattern analysis when only positive examples are reliable. SVDD is usually trained by solving a quadratic programming problem, which is time consuming. This paper formulates the Lagrangian of a simply modified SVDD model as a differentiable convex function over the nonnegative orthant. The resulting minimization problem can be solved by a simple iterative algorithm. The proposed algorithm is easy to implement, without requiring any particular optimization toolbox. Theoretical and experimental analysis show that the algorithm converges r-linearly to the unique minimum point. Extensive experiments on pattern classification were conducted, and compared to the quadratic programming based SVDD (QP-SVDD), the proposed approach is much more computationally efficient (hundreds of times faster) and yields similar performance in terms of receiver operating characteristic curve. Furthermore, the proposed method and QP-SVDD extract almost the same set of support vectors.

Keywords Support vector data description · Quadratic programming · Penalty function method · Lagrangian dual function · Support vectors

1 Introduction

There is a class of pattern recognition problems, such as novelty detection, where the task is to discriminate the pattern of interest from outliers. In such a situation, positive examples for training are relatively easier to obtain and more reliable. However, although negative examples are very abundant, it is usually difficult to sample enough useful negative examples for accurately modeling the outliers since they may belong to any class. In this case, it is reasonable to assume positive examples clustering in a certain way. As such, the goal is to accurately describe the class of positive examples as opposed to the wide range of negative examples.

For this purpose, Tax et al. [30, 31, 33] proposed a support vector data description (SVDD) method, which fits a tight hypersphere in the nonlinearly transformed feature space to include most of the positive examples. Thus, SVDD could be regarded as a description of the data distribution of interest. Extensive experiments [30, 31, 33] showed that SVDD is able to correctly identify negative examples in testing even though it has not seen any during training.

Like support vector machine (SVM) [34, Chap. 10], SVDD is a kernel based method, possessing all the related advantages of kernel machines. SVDD has been applied to various problems, including image classification [38], handwritten digit recognition [32], face recognition [18], remote sensing image analysis [22], medical image analysis [29], and multiclass problems [17, 37], to name a few. In addition, SVDD is a preliminary step for support vector clustering [4].

The formulation of SVDD leads us to a quadratic programming problem. Although decomposition techniques [24, 25] or sequential minimization method [26] could be employed to solve the quadratic programming, the training of SVDD has time complexity roughly of order $O(n^3)$, where n is the training set size (see Sects. 4.2 and 4.3 for experimental verification). Thus, training an SVDD model could be very expensive for large dataset. As such, given the wide application of SVDD, it is highly desirable to develop a time-efficient yet accurate enough training algorithm for SVDD.

In this paper, we first slightly modify the formulation of SVDD model, resulting in a more convex minimization problem with simpler constraints. We then apply the quadratic penalty function method [28, Sect. 6.2.2] from

✉ Songfeng Zheng
SongfengZheng@MissouriState.edu

1 Department of Mathematics, Missouri State University, Springfield, MO 65897, USA

optimization theory to absorb an equality constraint in the dual problem, obtaining a differentiable convex function over the nonnegative orthant as the approximated Lagrangian function, which can be efficiently minimized by a simple iterative algorithm. We thus call the proposed model as Lagrangian SVDD (L-SVDD). The proposed L-SVDD algorithm is easy to implement, requiring no particular optimization toolbox besides basic standard matrix operations. Theoretical and experimental analysis show that the algorithm converges r-linearly to the global minimum point and the algorithm has computational complexity of the order $O(n^2)$ multiplying the iteration number.

We test the proposed approach on face detection and handwritten digit recognition problems, and detailed performance measure comparison demonstrates that L-SVDD often yields testing accuracy very close to or slightly better than that of the Quadratic Programming based SVDD (QP-SVDD). More importantly, L-SVDD is much more computationally efficient than QP-SVDD (i.e., 200–400 times faster on the considered experiments). Furthermore, the two methods extract almost identically the same set of support vectors.

The following are several words about our notations. All scalars are represented by symbols (i.e., English or Greek letters) with normal font. All vectors will be denoted by **bold** lower case symbols, and all are column vectors unless transposed to a row vector by a prime superscript $'$. All matrices will be denoted by **bold** upper case symbols. For a vector $\mathbf{x}$ in $\mathbb{R}^n$, the plus function $\mathbf{x}_+$ is defined as $(\mathbf{x}_+)_i = \max\{0, x_i\}$, for $i = 1, \ldots, n$. For two vectors $\mathbf{a}$ and $\mathbf{b}$ in $\mathbb{R}^n$, $\mathbf{a} \geq \mathbf{b}$ means $a_i \geq b_i$ for each $i = 1, \ldots, n$. The notation $\mathbf{a} \perp \mathbf{b}$ means the two vectors $\mathbf{a}$ and $\mathbf{b}$ are perpendicular, that is, $a_1b_1 + a_2b_2 + \cdots + a_nb_n = 0$. For a vector $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|$ stands for its 2-norm, that is, $\|\mathbf{x}\| = \sqrt{x_1^2 + \cdots + x_n^2}$. For a square matrix $\mathbf{A}$ of size $n \times n$, $\|\mathbf{A}\|$ represent the matrix norm, that is

$$\|\mathbf{A}\| = \sup_{\mathbf{x} \neq \mathbf{0}_n} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}.$$

Thus, $\|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{x}\|$. If $\mathbf{A}$ is positive definite matrix, $\|\mathbf{A}\|$ is just the largest eigenvalue of $\mathbf{A}$.

The rest of this paper is organized as follows: Sect. 2 briefly reviews the formulation of SVDD and presents a simple modification to the original SVDD model; Sect. 3 formulates the approximated Lagrangian dual problem and proposes a simple iterative algorithm to solve it, and the convergence properties of the algorithm will also be investigated; Sect. 3 discusses the feasibility of two alternative schemes as well; Sect. 4 compares the performance measures in terms of receiver operating characteristic curve and training time of the proposed L-SVDD algorithm to those of QP-SVDD on two publicly available real-world datasets, and we also compare the support vectors the two methods extracted; the experimental results in Sect. 4 also verify the role of each parameter in the convergence behavior of the algorithm, and the computational complexity is verified by the experimental results as well; finally, Sect. 5 summarizes this paper and discusses some possible future research directions.

2 Support vector data description

Given training data $\{\mathbf{x}_i, i = 1, \ldots, n\}$ with the feature vector $\mathbf{x}_i \in \mathbb{R}^p$, let $\Phi(\cdot)$ be a nonlinear transformation[1] which maps the original data vector into a high dimensional Hilbert feature space $\mathcal{H}$. SVDD is looking for a hypersphere in $\mathcal{H}$, with radius $R > 0$ and center $\mathbf{c}$, which has a minimum volume containing most of the data. Therefore, we have to minimize R^2 constrained to $\|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq R^2$, for $i = 1, \ldots, n$. In addition, since the training sample might contain outliers, we can introduce a set of slack variables $\xi_i \geq 0$, as in the framework of support vector machine (SVM) [34, Chap. 10]. The slack variable ξ_i measures how much the squared distance from the training example $\mathbf{x}_i$ to the center $\mathbf{c}$ exceeds the radius squared. Therefore, the slack variable could be understood as a measure of errors.

Taking all the considerations into account, the SVDD model can be obtained by solving the following optimization problem

$$\min_{R,\mathbf{c},\boldsymbol{\xi}} \quad F(R, \mathbf{c}, \boldsymbol{\xi}) = R^2 + C \sum_{i=1}^{n} \xi_i, \tag{1}$$

with constraints

$$\|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq R^2 + \xi_i, \; \xi_i \geq 0, \quad \text{for } i = 1, \ldots, n, \tag{2}$$

where $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)'$ is the vector of slack variables, and the parameter $C > 0$ controls the tradeoff between the volume of the hypersphere and the permitted errors.

The Lagrangian dual of the above optimization problem is (refer to [30, 31, 33] for detailed derivations)

$$\min_{\boldsymbol{\alpha}} L(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}_i), \tag{3}$$

with constraints

$$\sum_{i=1}^{n} \alpha_i = 1, \quad 0 \leq \alpha_i \leq C \quad \text{for} \quad i = 1, \ldots, n, \tag{4}$$

[1] In SVM and SVDD literature, the explicit form of the function $\Phi(\cdot)$ is not important, and in fact, it is often difficult to write out $\Phi(\cdot)$ explicitly. What is important is the kernel function, that is, the inner product of $\Phi(\mathbf{x}_i)$ and $\Phi(\mathbf{x}_j)$.

where $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)'\Phi(x_j)$ is the kernel function which satisfies Mercer's condition [34, Chap. 10], and $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)'$ with α_i being the Lagrangian multiplier for the i-th constraint in Eq. (2).

Similar to the works on SVM [21] and support vector regression [23], we consider the sum of squared errors in the objective function given in Eq. (1), that is, we modify Eq. (1) to

$$\min_{R,\mathbf{c},\boldsymbol{\xi}} \quad \tilde{F}(R, \mathbf{c}, \boldsymbol{\xi}) = R^2 + C\sum_{i=1}^{n} \xi_i^2. \tag{5}$$

With this slight modification, the objective function becomes more convex because of the square terms. Furthermore, the nonnegativity constraint in Eq. (2) could be removed, which is proved in the following.

Proposition 1 *If $(\hat{R}, \hat{\mathbf{c}}, \hat{\boldsymbol{\xi}})$ is the minimum point of function $\tilde{F}(R, \mathbf{c}, \boldsymbol{\xi})$ defined in Eq. (5), with the constraints $\|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 \le R^2 + \xi_i$, for $i = 1, \dots, n$, then all components of $\hat{\boldsymbol{\xi}}$ should be nonnegative.*

Proof Assume $\hat{\boldsymbol{\xi}} = (\hat{\xi}_1, \hat{\xi}_2, \dots, \hat{\xi}_n)'$ and without loss of generality, assume $\hat{\xi}_1 < 0$. Let $\tilde{\boldsymbol{\xi}} = (0, \hat{\xi}_2, \dots, \hat{\xi}_n)'$, that is, we replace the first component of $\hat{\boldsymbol{\xi}}$ (which is negative) by 0 and keep others unchanged. Since $\hat{\xi}_1$ satisfies the constraint $\|\Phi(\mathbf{x}_1) - \hat{\mathbf{c}}\|^2 \le \hat{R}^2 + \hat{\xi}_1$, we must have $\|\Phi(\mathbf{x}_1) - \hat{\mathbf{c}}\|^2 \le \hat{R}^2 + 0$ since $\hat{\xi}_1 < 0$. By assumption, the constraints are satisfied at $\hat{\xi}_2, \dots, \hat{\xi}_n$. Hence, the constraints are satisfied at all components of $\tilde{\boldsymbol{\xi}}$.

However, there is

$$\tilde{F}(\hat{R}, \hat{\mathbf{c}}, \tilde{\boldsymbol{\xi}}) = \hat{R}^2 + C\sum_{i=2}^{n} \hat{\xi}_i^2 < \hat{R}^2 + C\sum_{i=1}^{n} \hat{\xi}_i^2 = \tilde{F}(\hat{R}, \hat{\mathbf{c}}, \hat{\boldsymbol{\xi}})$$

since $\hat{\xi}_1 < 0$ by assumption, and this is contradiction to the assumption that $(\hat{R}, \hat{\mathbf{c}}, \hat{\boldsymbol{\xi}})$ is the minimum point. Thus, at the minimum point, there must be $\hat{\xi}_1 \ge 0$. In the same manner, it can be argued that all components of $\hat{\boldsymbol{\xi}}$ should be nonnegative. Consequently, the nonnegative constraint on $\boldsymbol{\xi}$ is not necessary, thus can be removed. □

The Lagrangian function of this new problem is

$$\tilde{L}(R, \mathbf{c}, \boldsymbol{\xi}, \boldsymbol{\alpha}) = R^2 + C\sum_{i=1}^{n} \xi_i^2 + \sum_{i=1}^{n} \alpha_i \left[\|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 - R^2 - \xi_i\right],$$

with α_i's being the nonnegative Lagrangian multipliers. At the optimal point, the partial derivative to the primal variables are zeros. Thus, there are

$$\frac{\partial \tilde{L}}{\partial R} = 0 \Rightarrow 2R - \sum_{i=1}^{n} 2R\alpha_i = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i = 1, \tag{6}$$

$$\frac{\partial \tilde{L}}{\partial \xi_i} = 0 \Rightarrow 2C\xi_i - \alpha_i = 0 \Rightarrow \xi_i = \frac{\alpha_i}{2C} \quad \text{for} \quad i = 1, 2, \dots, n,$$

and

$$\frac{\partial \tilde{L}}{\partial \mathbf{c}} = 0 \Rightarrow -2\sum_{i=1}^{n} \alpha_i(\Phi(\mathbf{x}_i) - \mathbf{c}) = 0 \Rightarrow \mathbf{c} = \sum_{i=1}^{n} \alpha_i \Phi(\mathbf{x}_i).$$

Substituting these results to the Lagrangian function, we have the dual function as

$$\tilde{L}(\boldsymbol{\alpha}) = -\frac{1}{4C}\sum_{i=1}^{n} \alpha_i^2 - \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}_i).$$

The purpose is now to maximize $\tilde{L}(\boldsymbol{\alpha})$ or minimize $L(\boldsymbol{\alpha}) = -\tilde{L}(\boldsymbol{\alpha})$[2] with respect to nonnegative α_i's with the constraint in Eq. (6), that is

$$\min_{\boldsymbol{\alpha}} \quad L(\boldsymbol{\alpha}) = \frac{1}{4C}\sum_{i=1}^{n} \alpha_i^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}_i), \tag{7}$$

with constraints

$$\sum_{i=1}^{n} \alpha_i = 1, \quad \alpha_i \ge 0 \quad \text{for} \quad i = 1, \dots, n. \tag{8}$$

With the sum of errors replaced by sum of squared errors, the resulting dual problem in Eq. (7) has an extra quadratic term, compared to the original dual problem in Eq. (3), and this will improve the convexity of the objective function. Moreover, by comparing the constraints in Eqs. (4) and (8), it is clear that the new optimization problem has simpler constraints, without any upper bound for the dual variables.

As implemented in popular SVM toolboxes [7, 14, 15], the quadratic programming problems in Eqs. (3) and (7) can be solved by decomposition methods [24, 25] or sequential minimal optimization method [26]. However, these algorithms are computationally expensive with time complexity roughly $O(n^3)$. Thus, a fast training algorithm for SVDD which can achieve similar accuracy as the quadratic programming method is highly appreciated.

[2] We slightly abuse the notation here because $L(\boldsymbol{\alpha})$ was used in Eq. (3). However, this will not cause any confusion because all of our following discussions are based on Eq. (7).

3 Lagrangian support vector data description

3.1 The algorithm

Let $\mathbf{K}$ be the $n \times n$ kernel matrix, that is, $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, let vector $\mathbf{u}$ be formed by the diagonal elements of kernel matrix $\mathbf{K}$, let $\mathbf{I}_n$ be the $n \times n$ identity matrix, and let $\mathbf{1}_n$ ($\mathbf{0}_n$) be the n-dimensional vector of all 1's (0's). The optimization problem in Eq. (7) could be written compactly in matrix form as

$$\min_{\boldsymbol{\alpha}} \quad L(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}'\left(\frac{\mathbf{I}_n}{2C} + 2\mathbf{K}\right)\boldsymbol{\alpha} - \mathbf{u}'\boldsymbol{\alpha}, \tag{9}$$

with constraints

$$\mathbf{1}_n'\boldsymbol{\alpha} = 1 \quad \text{and} \quad \boldsymbol{\alpha} \geq \mathbf{0}_n. \tag{10}$$

To deal with the equality constraint in Eq. (10), we consider the penalty function method [28, Sect. 6.2.2] from optimization theory. The basic idea is integrating the original objective function with a function which incorporates some constraints, in order to approximate a constrained optimization problem by an unconstrained problem or one with simpler constraints. For our problem, we consider the following function

$$\begin{aligned} f(\boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{\alpha}'\left(\frac{\mathbf{I}_n}{2C} + 2\mathbf{K}\right)\boldsymbol{\alpha} - \mathbf{u}'\boldsymbol{\alpha} + \rho(\mathbf{1}_n'\boldsymbol{\alpha} - 1)^2 \\ &= \frac{1}{2}\boldsymbol{\alpha}'\left(\frac{\mathbf{I}_n}{2C} + 2\mathbf{K} + 2\rho\mathbf{J}_n\right)\boldsymbol{\alpha} - (\mathbf{u} + 2\rho\mathbf{1}_n)'\boldsymbol{\alpha} + \rho, \end{aligned} \tag{11}$$

where $\mathbf{J}_n$ is the $n \times n$ matrix of all elements being 1. As proved in [28], as the penalty parameter $\rho \to \infty$, the minimum point of Eq. (11) with $\boldsymbol{\alpha} \geq \mathbf{0}_n$ converges to the solution to Eq. (9) with constraints in Eq. (10).

Let

$$\mathbf{Q} = \frac{\mathbf{I}_n}{2C} + 2\mathbf{K} + 2\rho\mathbf{J}_n \quad \text{and} \quad \mathbf{v} = \mathbf{u} + 2\rho\mathbf{1}_n. \tag{12}$$

The matrix $\mathbf{Q}$ is positive definite because both $\mathbf{K}$ and $\mathbf{J}_n$ are positive semi-definite while $\mathbf{I}_n$ is positive definite. Ignoring the constant term in Eq. (11), we can formulate the approximated minimization problem as

$$\min_{\boldsymbol{\alpha} \geq \mathbf{0}_n} \quad \frac{1}{2}\boldsymbol{\alpha}'\mathbf{Q}\boldsymbol{\alpha} - \mathbf{v}'\boldsymbol{\alpha}. \tag{13}$$

The Kuhn–Tucker stationary-point problem [20, p. 94, KTP 7.2.4] for Eq. (13) is

$$\mathbf{Q}\boldsymbol{\alpha} - \mathbf{v} - \mathbf{w} = \mathbf{0}_n, \ \mathbf{w}'\boldsymbol{\alpha} = 0, \ \boldsymbol{\alpha} \geq \mathbf{0}_n, \ \mathbf{w} \geq \mathbf{0}_n.$$

From these equations, we have

$$\mathbf{w} = \mathbf{Q}\boldsymbol{\alpha} - \mathbf{v} \geq \mathbf{0}_n \quad \text{and} \quad (\mathbf{Q}\boldsymbol{\alpha} - \mathbf{v})'\boldsymbol{\alpha} = 0,$$

which can be summarized as solving the classical linear complementarity problem [10], that is, solving for $\boldsymbol{\alpha}$, such that,

$$\mathbf{0}_n \leq (\mathbf{Q}\boldsymbol{\alpha} - \mathbf{v}) \perp \boldsymbol{\alpha} \geq \mathbf{0}_n. \tag{14}$$

Since the matrix $\mathbf{Q}$ is symmetric positive-definite, the existence and uniqueness of the solution to Eq. (14) is guaranteed [9]. The optimality condition in Eq. (14) is satisfied if and only if for any $\gamma > 0$, the relationship

$$\mathbf{Q}\boldsymbol{\alpha} - \mathbf{v} = (\mathbf{Q}\boldsymbol{\alpha} - \mathbf{v} - \gamma\boldsymbol{\alpha})_+ \tag{15}$$

holds. See Appendix for a proof.

To obtain a solution to the above problem, we start from an initial point $\boldsymbol{\alpha}^0$, and apply the following iterative scheme

$$\boldsymbol{\alpha}^{k+1} = \mathbf{Q}^{-1}(\mathbf{v} + (\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v} - \gamma\boldsymbol{\alpha}^k)_+). \tag{16}$$

The initial point $\boldsymbol{\alpha}^0$ could be any vector, but in our implementation, we take $\boldsymbol{\alpha}^0 = \mathbf{Q}^{-1}\mathbf{v}$. We summarize the algorithm for L-SVDD as in Algorithm 1 below, and the convergence analysis will be given in Sect. 3.2.

Algorithm 1: Lagrangian Support Vector Data Description

```
0. Initialization: choose the starting point as $\boldsymbol{\alpha}^0 = \mathbf{Q}^{-1}\mathbf{v}$,
   find $\boldsymbol{\alpha}^1$ using Eq. (16), set $k = 1$, set the iteration number
   as $M$, and set the error tolerance as $\boldsymbol{\varepsilon}$.
1. while $k < M$ and $||\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1}|| > \epsilon$ do:
2.     Set $k = k + 1$.
3.     Find $\boldsymbol{\alpha}^k$ using Eq. (16).
4. end while
5. Return the vector $\boldsymbol{\alpha}^k$.
```

Remark 1 In Algorithm 1, we terminate the program when the solution does not change too much. Since the purpose is to find a solution to Eq. (14), we can also terminate the program if the absolute value of inner product $(\boldsymbol{\alpha}^k)'(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v})$ is below a certain level. However, since it includes matrix and vector multiplication, this stopping criterion is more expensive to evaluate than the one in Algorithm 1. Thus, in our implementation, we choose to use the stopping rule given in Algorithm 1. Our experimental results show that when the algorithm stops, the inner product indeed is very close to 0. Please see Sects. 4.2 and 4.3 for the detailed results.

Remark 2 Each iteration of Algorithm 1 includes matrix multiplying vector, vector addition/subtraction, and taking positive part of a vector component-wise, among which the most expensive operation is matrix multiplying vector, which has computational complexity of order $O(n^2)$. We thus expect the computational complexity of Algorithm 1 to be about iteration number multiplying $O(n^2)$. Sections 4.2 and 4.3 verify this analysis experimentally.

Similar to SVM, we call the training examples with corresponding α_i's nonzero as support vectors. Once α_i's are obtained, the radius R can be computed from the set of support vectors [30, 31]. In the stage of decision making, if the distance from a new example $\mathbf{x}$ to the center is less than the radius R, it is classified as a positive example; otherwise, it is classified as a negative example. That is, the decision rule is

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sign}\left(R^2 - \left\|\Phi(\mathbf{x}) - \sum_{i=1}^{n} \alpha_i \Phi(\mathbf{x}_i)\right\|^2\right) \\ &= \operatorname{sign}\left(2\sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}) - K(\mathbf{x}, \mathbf{x}) + b\right), \end{aligned} \tag{17}$$

where $b = R^2 - \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$.

3.2 Convergence analysis

To analyze the convergence property of Algorithm 1, we need the following

Lemma 1 *Let* $\mathbf{a}$ *and* $\mathbf{b}$ *be two points in* $\mathbb{R}^p$, *then*

$$\|\mathbf{a}_+ - \mathbf{b}_+\| \le \|\mathbf{a} - \mathbf{b}\|. \tag{18}$$

Proof For two real numbers a and b, there are four situations

(1) $a \ge 0$ and $b \ge 0$, then $|a_+ - b_+| = |a - b|$;
(2) $a \ge 0$ and $b \le 0$, then $|a_+ - b_+| = |a - 0| \le |a - b|$;
(3) $a \le 0$ and $b \ge 0$, then $|a_+ - b_+| = |0 - b| \le |a - b|$;
(4) $a \le 0$ and $b \le 0$, then $|a_+ - b_+| = |0 - 0| \le |a - b|$.

In summary, for one dimensional case, there is $|a_+ - b_+|^2 \le |a - b|^2$.

Assume that Eq. (18) is true for p dimensional vectors $\mathbf{a}_p$ and $\mathbf{b}_p$. Denote the $p+1$ dimensional vectors $\mathbf{a}$ and $\mathbf{b}$ as[3]

$$\mathbf{a} = (\mathbf{a}_p, a_{p+1}) \quad \text{and} \quad \mathbf{b} = (\mathbf{b}_p, b_{p+1}),$$

where a_{p+1} and b_{p+1} are real numbers. Then,

$$\begin{aligned} \|\mathbf{a}_+ - \mathbf{b}_+\|^2 &= \|((\mathbf{a}_p)_+ - (\mathbf{b}_p)_+, (a_{p+1})_+ - (b_{p+1})_+)\|^2 \\ &= \|(\mathbf{a}_p)_+ - (\mathbf{b}_p)_+\|^2 + ((a_{p+1})_+ - (b_{p+1})_+)^2 \\ &\le \|\mathbf{a}_p - \mathbf{b}_p\|^2 + (a_{p+1} - b_{p+1})^2 = \|\mathbf{a} - \mathbf{b}\|^2, \end{aligned} \tag{19}$$

where in Eq. (19), we used the assumption on the p dimensional vectors, the special result for one dimensional case, and the definition of Euclidean norm.

By induction, Eq. (18) is proved. □

[3] For notational convenience, in this proof, we assume all the vectors are row vectors. Clearly, the result also applies to column vectors.

With the aid of Lemma 1, we are ready to study the convergence behavior of Algorithm 1, and we have the following conclusion.

Proposition 2 *With* $0 < \gamma < 1/C$, *the sequence* $\boldsymbol{\alpha}^k$ *obtained by Algorithm 1 converges r-linearly* [2] *to the unique solution* $\bar{\boldsymbol{\alpha}}$ *of Eq.* (13), *that is*

$$\limsup_{k\to\infty} \|\boldsymbol{\alpha}^k - \bar{\boldsymbol{\alpha}}\|^{1/k} < 1.$$

Proof The convexity of the objective function in Eq. (13) and the convexity of the feasible region ensure the existence and uniqueness of solution $\bar{\boldsymbol{\alpha}}$ to Eq. (13). Since $\bar{\boldsymbol{\alpha}}$ is the solution to Eq. (13), it must satisfy the optimality condition in Eq. (15), that is, for any $\gamma > 0$

$$\mathbf{Q}\bar{\boldsymbol{\alpha}} = \mathbf{v} + (\mathbf{Q}\bar{\boldsymbol{\alpha}} - \mathbf{v} - \gamma\bar{\boldsymbol{\alpha}})_+. \tag{20}$$

Multiplying Eq. (16) by $\mathbf{Q}$ and subtracting Eq. (20), and then taking norm gives us

$$\|\mathbf{Q}\boldsymbol{\alpha}^{k+1} - \mathbf{Q}\bar{\boldsymbol{\alpha}}\| = \|(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v} - \gamma\boldsymbol{\alpha}^k)_+ - (\mathbf{Q}\bar{\boldsymbol{\alpha}} - \mathbf{v} - \gamma\bar{\boldsymbol{\alpha}})_+\|. \tag{21}$$

Applying Lemma 1 to the vectors $\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v} - \gamma\boldsymbol{\alpha}^k$ and $\mathbf{Q}\bar{\boldsymbol{\alpha}} - \mathbf{v} - \gamma\bar{\boldsymbol{\alpha}}$ in Eq. (21), we have

$$\begin{aligned} \|\mathbf{Q}\boldsymbol{\alpha}^{k+1} - \mathbf{Q}\bar{\boldsymbol{\alpha}}\| &\le \|(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v} - \gamma\boldsymbol{\alpha}^k) - (\mathbf{Q}\bar{\boldsymbol{\alpha}} - \mathbf{v} - \gamma\bar{\boldsymbol{\alpha}})\| \\ &= \|(\mathbf{Q} - \gamma\mathbf{I})(\boldsymbol{\alpha}^k - \bar{\boldsymbol{\alpha}})\| \\ &= \|(\mathbf{I} - \gamma\mathbf{Q}^{-1})(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{Q}\bar{\boldsymbol{\alpha}})\| \\ &\le \|\mathbf{I} - \gamma\mathbf{Q}^{-1}\| \cdot \|\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{Q}\bar{\boldsymbol{\alpha}}\|. \end{aligned} \tag{22}$$

In the definition of $\mathbf{Q}$ in Eq. (12), it is clear that the matrix $2\mathbf{K} + 2\rho\mathbf{J}_n$ is positive semi-definite because both $\mathbf{K}$ and $\mathbf{J}_n$ are, and we denote its eigenvalues as λ_i, $i = 1, 2, \ldots, n$. Then the eigenvalues of $\mathbf{Q}$ are $\frac{1}{2C} + \lambda_i$ and the eigenvalues of $\mathbf{Q}^{-1}$ are $(\frac{1}{2C} + \lambda_i)^{-1}$, $i = 1, 2, \ldots, n$. To make the sequence $\|\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{Q}\bar{\boldsymbol{\alpha}}\|$ converge, Eq. (22) indicates that we need $\|\mathbf{I} - \gamma\mathbf{Q}^{-1}\| < 1$, that is, the eigenvalues of $\mathbf{I} - \gamma\mathbf{Q}^{-1}$ are all between -1 and 1,

$$-1 < 1 - \gamma\left(\frac{1}{2C} + \lambda_i\right)^{-1} < 1 \quad \text{for} \quad i = 1, 2, \ldots, n,$$

or

$$0 < \gamma < 2\left(\frac{1}{2C} + \lambda_i\right) = \frac{1}{C} + 2\lambda_i \quad \text{for} \quad i = 1, 2, \ldots, n.$$

Thus, with the choice of $0 < \gamma < 1/C$, we have

$$c = \|\mathbf{I} - \gamma\mathbf{Q}^{-1}\| < 1.$$

Recursively applying Eq. (22), we have that for any k,

$$\|\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{Q}\bar{\boldsymbol{\alpha}}\| \le c^k \|\mathbf{Q}\boldsymbol{\alpha}^0 - \mathbf{Q}\bar{\boldsymbol{\alpha}}\|.$$

Consequently,

$$\begin{aligned}\|\boldsymbol{\alpha}^k - \bar{\boldsymbol{\alpha}}\| &= \|\mathbf{Q}^{-1}\mathbf{Q}(\boldsymbol{\alpha}^k - \bar{\boldsymbol{\alpha}})\| \le \|\mathbf{Q}^{-1}\|\|\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{Q}\bar{\boldsymbol{\alpha}}\| \\ &\le c^k\|\mathbf{Q}^{-1}\|\|\mathbf{Q}\boldsymbol{\alpha}^0 - \mathbf{Q}\bar{\boldsymbol{\alpha}}\| = Ac^k,\end{aligned} \tag{23}$$

where $A = \|\mathbf{Q}^{-1}\|\|\mathbf{Q}\boldsymbol{\alpha}^0 - \mathbf{Q}\bar{\boldsymbol{\alpha}}\| > 0$. Hence,

$$\limsup_{k\to\infty} \|\boldsymbol{\alpha}^k - \bar{\boldsymbol{\alpha}}\|^{1/k} \le \limsup_{k\to\infty} A^{1/k}c = c < 1.$$

This proves the proposition. □

Remark 3 The proof of Proposition 2 enables us to estimate the iteration number M. If we require the accuracy of the solution to be ε, in the sense that $\|\boldsymbol{\alpha}^M - \bar{\boldsymbol{\alpha}}\| < \varepsilon$. From Eq. (23), it is sufficient to have $\|\boldsymbol{\alpha}^M - \bar{\boldsymbol{\alpha}}\| < Ac^M = \varepsilon$, where $A = \|\mathbf{Q}^{-1}\|\|\mathbf{Q}\boldsymbol{\alpha}^0 - \mathbf{Q}\bar{\boldsymbol{\alpha}}\|$ and $c = \|\mathbf{I} - \gamma\mathbf{Q}^{-1}\|$. This enables us to solve for M as

$$M = \frac{\log\varepsilon - \log A}{\log c}.$$

However, A cannot be calculated because $\bar{\boldsymbol{\alpha}}$ is unknown. Hence, in the implementation, we set M as a large number and terminate the program according to the criterion in Algorithm 1.

From the proof of Proposition 2, the convergence rate of Algorithm 1 depends on c, the norm of matrix $\mathbf{I} - \gamma\mathbf{Q}^{-1}$. The analysis in the proof shows that a smaller value of c gives a faster convergence rate. By the definition of matrix norm, there is

$$c = \|\mathbf{I} - \gamma\mathbf{Q}^{-1}\| = \max_i \left\{1 - \gamma\left(\frac{1}{2C} + \lambda_i\right)^{-1}\right\},$$

from which it is clear that a larger value of γ makes c smaller and consequently makes the algorithm converge faster.

In accord with Proposition 2, let us assume that $\gamma = a/C$ for some constant $0 < a < 1$. We have,

$$\begin{aligned} c &= \max_i \left\{1 - \gamma\left(\frac{1}{2C} + \lambda_i\right)^{-1}\right\} \\ &= \max_i \left\{1 - \frac{a}{C}\left(\frac{1}{2C} + \lambda_i\right)^{-1}\right\} \\ &= \max_i \left\{1 - \left(\frac{1}{2a} + \lambda_i\frac{C}{a}\right)^{-1}\right\} \\ &= \max_i \left\{1 - \frac{2a}{1+2C\lambda_i}\right\}.\end{aligned}$$

Thus, a small value of C and the smallest eigenvalue will make $\frac{2a}{1+2C\lambda_i}$ large hence c small, and consequently the algorithm will converge faster. However, to the best of our knowledge, there is no theoretical conclusion about the dependence between the eigenvalues of $2\mathbf{K} + 2\rho\mathbf{J}_n$ and ρ. Fortunately, our numerical tests revealed that with the change of ρ, the largest eigenvalue of $2\mathbf{K} + 2\rho\mathbf{J}_n$ changes dramatically while the smallest eigenvalue does not change too much. Since c depends on the smallest eigenvalue of $2\mathbf{K} + 2\rho\mathbf{J}_n$, we thus conclude that the convergence rate of Algorithm 1 is not affected much by ρ.

In summary, we reach the conclusion that, to achieve a faster convergence rate, we should set γ large and C small (γ depends on C in our implementation), and ρ does not significantly impact the convergence behavior. We should mention that C also controls the error of the model, so setting C small might make the resulting model perform poorly in classification. We will numerically verify these analysis in Sect. 4.2.

3.3 Discussion on two alternatives

We train the L-SVDD model by solving the linear complementarity problem in Eq. (14), and Algorithm 1 is based on the condition in Eq. (15) at the optimum point. Alternatively, the optimality condition can be written as

$$\boldsymbol{\alpha} = (\boldsymbol{\alpha} - \tilde{\gamma}(\mathbf{Q}\boldsymbol{\alpha} - \mathbf{v}))_+, \tag{24}$$

where $\tilde{\gamma} > 0$.

In principle, similar to Algorithm 1, we can design an algorithm based on recursive relation

$$\boldsymbol{\alpha}^{k+1} = (\boldsymbol{\alpha}^k - \tilde{\gamma}(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v}))_+, \tag{25}$$

with some appropriately selected $\tilde{\gamma}$. Intuitively, the algorithm based on Eq. (25) should be more computationally efficient in each iteration than Algorithm 1 which is based on Eq. (16). The reason is that Eq. (16) involves three vector addition/subtraction operations and two matrix and vector multiplications, while Eq. (25) only includes two vector addition/subtraction operations and one matrix and vector multiplication.

To choose $\tilde{\gamma}$, as in the proof of Proposition 2, we let the unique solution to Eq. (14) be $\bar{\boldsymbol{\alpha}}$, which must satisfy

$$\bar{\boldsymbol{\alpha}} = (\bar{\boldsymbol{\alpha}} - \tilde{\gamma}(\mathbf{Q}\bar{\boldsymbol{\alpha}} - \mathbf{v}))_+. \tag{26}$$

Subtracting Eq. (26) from Eq. (25), taking norm, and applying Lemma 1, we have

$$\begin{aligned}\|\boldsymbol{\alpha}^{k+1} - \bar{\boldsymbol{\alpha}}\| &= \|(\boldsymbol{\alpha}^k - \tilde{\gamma}(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v}))_+ - (\bar{\boldsymbol{\alpha}} - \tilde{\gamma}(\mathbf{Q}\bar{\boldsymbol{\alpha}} - \mathbf{v}))_+\| \\ &\le \|(\boldsymbol{\alpha}^k - \tilde{\gamma}(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v})) - (\bar{\boldsymbol{\alpha}} - \tilde{\gamma}(\mathbf{Q}\bar{\boldsymbol{\alpha}} - \mathbf{v}))\| \\ &= \|(\mathbf{I} - \tilde{\gamma}\mathbf{Q})(\boldsymbol{\alpha}^k - \bar{\boldsymbol{\alpha}})\| \le \|\mathbf{I} - \tilde{\gamma}\mathbf{Q}\| \cdot \|\boldsymbol{\alpha}^k - \bar{\boldsymbol{\alpha}}\|.\end{aligned}$$

Thus, to make the potential algorithm converge, we must have $\|\mathbf{I} - \tilde{\gamma}\mathbf{Q}\| < 1$.

Denote the eigenvalues of matrix $2\mathbf{K}+2\rho\mathbf{J}_n$ as λ_i, $i=1,2,\dots,n$, then the eigenvalues of $\mathbf{I}-\tilde{\gamma}\mathbf{Q}$ are $1-\tilde{\gamma}(\frac{1}{2C}+\lambda_i)$. To ensure $\|\mathbf{I}-\tilde{\gamma}\mathbf{Q}\|<1$, we need all the eigenvalues of $\mathbf{I}-\tilde{\gamma}\mathbf{Q}$ to be between -1 and 1, that is

$$-1<1-\tilde{\gamma}\left(\frac{1}{2C}+\lambda_i\right)<1 \quad \text{for} \quad i=1,2,\dots,n,$$

or

$$0<\tilde{\gamma}<\frac{2}{\frac{1}{2C}+\lambda_i}=\frac{4C}{1+2C\lambda_i} \quad \text{for} \quad i=1,2,\dots,n.$$

Thus, we should choose $\tilde{\gamma}$ as

$$0<\tilde{\gamma}<\frac{4C}{1+2C\lambda_{\max}},$$

where $\lambda_{\max}=\max\{\lambda_1,\lambda_2,\dots,\lambda_n\}$.

However, our experimental results[4] show that $\lambda_{\max}$ is large because ρ need to be large, as required by the penalty function method. As a result, this will make $\tilde{\gamma}$ very small and consequently, make the potential algorithm based on Eq. (25) converge slowly. We tested this alternative idea in our experiments, and the result showed that, under the same stopping criterion, compared to Algorithm 1, although each iteration is more time efficient, the algorithm based on Eq. (25) needs much more iterations to converge,[5] hence spending more time.

The purpose of Algorithm 1 is to solve the quadratic minimization problem given in Eq. (13), and another alternative is to apply interior point method (IPM) [2, 5, 28] to this problem. Let $\mathbf{z}_k$ be the k-th step solution to the problem of minimizing some convex function $g(\mathbf{z})$ with some constraints using IPM, and denote the global minimum point as $\mathbf{z}_\mu$. In [2], it was proved that $\mathbf{z}_k$ converges to $\mathbf{z}_\mu$ not only r-linearly, but also q-superlinearly in the sense that $\|\mathbf{z}_{k+1}-\mathbf{z}_\mu\|/\|\mathbf{z}_k-\mathbf{z}_\mu\|\to 0$. Proposition 2 shows that the proposed L-SVDD algorithm also has r-linear convergence rate. However, L-SVDD algorithm cannot achieve q-superlinear convergence rate. This means that theoretically, IPM should converge in fewer iterations than L-SVDD.

In each iteration of L-SVDD algorithm, the operation is quite simple, with the most expensive computation being matrix and vector multiplication. However, each IPM iteration is much more complicated, because it includes evaluating the objective function and constraint values, calculating the gradients and Hessian, finding the search direction, and conducting a backtracking line search to update the solution. These operations include several matrix inversion and more matrix multiplications. Thus, each iteration of IPM is much more expensive than L-SVDD. Hence, although IPM converges in fewer iterations, it might spend more computing time and resource (e.g., memory) than L-SVDD.

We developed the interior point method based SVDD model (IPM-SVDD) by adapting the MATLAB code from https://pcarbo.github.io/convexprog.html. Section 4.2 presents the comparison between IPM-SVDD and L-SVDD in terms of iteration number and CPU time for achieving convergence.

4 Experimental results and analysis

On a face dataset and the USPS handwritten digit dataset, we compared the performances of the proposed Lagrangian SVDD (L-SVDD) and the ordinary Quadratic Programming based SVDD (QP-SVDD), which is obtained by applying a quadratic programming solver to Eq. (7) with constraints in Eq. (8).

4.1 Experiment setup and performance measures

The program for L-SVDD was developed using MATLAB, and we did not do any specific code optimization; QP-SVDD was implemented based on the MATLAB SVM toolbox [14] with the core quadratic programming solver written in C++. The source code of this work is available upon request. All the experiments were conducted on a laptop computer with Intel(R) Core(TM) i5-2450M CPU 2.50 GHz and 4 GB memory, with Windows 7 Professional operating system and MATLAB® R2007b as the platform. During all experiments that incorporated measurement of running time, one core was used solely for the experiments, and the number of other processes running on the system was minimized.

In our experiments, we adopted the Gaussian kernel

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

with $\sigma=8$. We set the SVDD control parameter $C=2$ in the algorithms. The parameter setting in our experiments might not be optimal to achieve the best testing performance. Nonetheless, our purpose is not to achieve the least testing error, but to compare the performances between L-SVDD and QP-SVDD; therefore, the comparison is fair as long as the parameter settings are the same for the two algorithms. In general, we can select the optimal parameter setting (C,σ) by applying cross validation, generalized approximate cross validation [36], or other criteria mentioned in [8, 13], but

[4] To the best of our knowledge, there is no theoretical result regarding the dependence between the largest eigenvalue of matrix $2\mathbf{K}+2\rho\mathbf{J}_n$ and the parameter ρ.

[5] For instance, on the face detection problem in Sect. 4.2, to achieve the error tolerance of 10^{-5}, the algorithm based on Eq. (25) needs more than 20,000 iterations to converge.

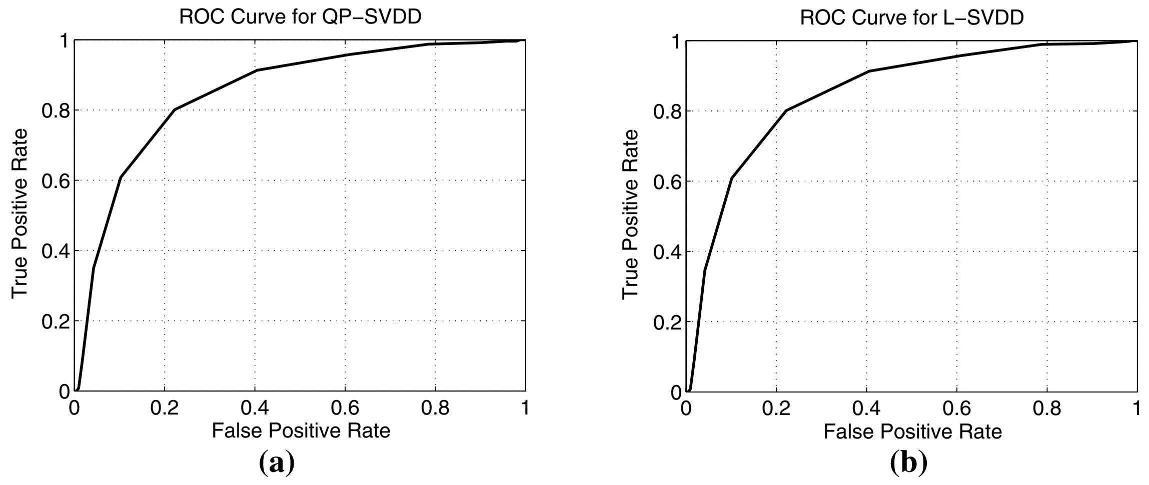

Fig. 1 The ROC curves of **a** QP-SVDD and **b** L-SVDD on CBCL testing dataset

since it is not the focus of this paper, we choose not to pursue further in this issue.

To make the L-SVDD algorithm converge fast, from the conclusion at the end of Sect. 3.2, we should set γ a large value. In all of our experiments, we chose $\gamma = 0.95/C$ to ensure the convergence of Algorithm 1. According to the theoretical property of penalty function method [28, Sect. 6.2.2], the parameter ρ in the definition of $\mathbf{Q}$ in Eq. (12) should be as large as possible. In our experiments, we found that setting $\rho = 200$ is enough to ensure the closeness of $\mathbf{1}_n'\boldsymbol{\alpha}$ and 1 (in fact, all the experimental results have $|\mathbf{1}_n'\boldsymbol{\alpha} - 1| < 0.002$ when the L-SVDD training algorithm terminates). Section 4.2 also studies the roles of the parameters C, γ, and ρ in the convergence behavior of Algorithm 1. The iteration number M in Algorithm 1 was set to be 3000, although in our experiments we found that most times the algorithm converges in 1000 iterations. The tolerance parameter in Algorithm 1 was set to be 10^{-5}.

In the decision rule given by Eq. (17), we changed the value of the parameter b, and for each value of b, we calculated the true positive rate and false positive rate. We then plot the true positive rate vs. the false positive rate, resulting the receiver operating characteristic (ROC) curve [11], which will be used to illustrate the performance of the classifier. The classifier performs better if the corresponding ROC curve is higher. In order to numerically compare the ROC curves of different methods, we calculate the area under the curve (AUC) [11]. The larger the AUC, the better the overall performance of the classifier.

To compare the support vectors extracted by the two algorithms, we regard the support vectors given by QP-SVDD as true support vectors, and denote them as the set SV_Q, denote the support vectors from L-SVDD as SV_L. We define precision and recall [27] as

$$\text{precision} = \frac{|SV_Q \cap SV_L|}{|SV_L|} \quad \text{and} \quad \text{recall} = \frac{|SV_Q \cap SV_L|}{|SV_Q|},$$

where $|S|$ represents the size of a set S. High precision means that L-SVDD finds substantially more correct support vectors than incorrect ones, while high recall means that L-SVDD extracts most of the correct support vectors.

4.2 Face detection

This experiment used the face dataset provided by the Center for Biological and Computational Learning (CBCL) at MIT. The training set consists of 6977 images, with 2429 face images and 4548 non-face images; the testing set consists of 24,045 images, including 472 face images and 23,573 non-face images. Each image is of size 19×19, with pixel values between 0 and 1. We did not extract any specific features for face detection (e.g., the features used in [35]), instead, we directly used the pixel values as input to L-SVDD and QP-SVDD.

Figure 1 shows the ROC curves of QP-SVDD and L-SVDD on the testing set. The ROC curves are so close that if we plot them in the same figure, they will be indistinguishable. The closeness of the ROC curves indicates that the two resulting classifiers should perform very closely. Indeed, the AUC for QP-SVDD is 0.8506 while the AUC for L-SVDD is 0.8512, that is, L-SVDD classifier performs even slightly better. On our computer, QP-SVDD spent 7301.8 s in training while it took the L-SVDD training algorithm 862 iterations to converge, consuming only 13.0729 s. In another word, the proposed L-SVDD is almost 560 times faster than QP-SVDD in training. Thus, we could conclude that L-SVDD is much more time-efficient in training

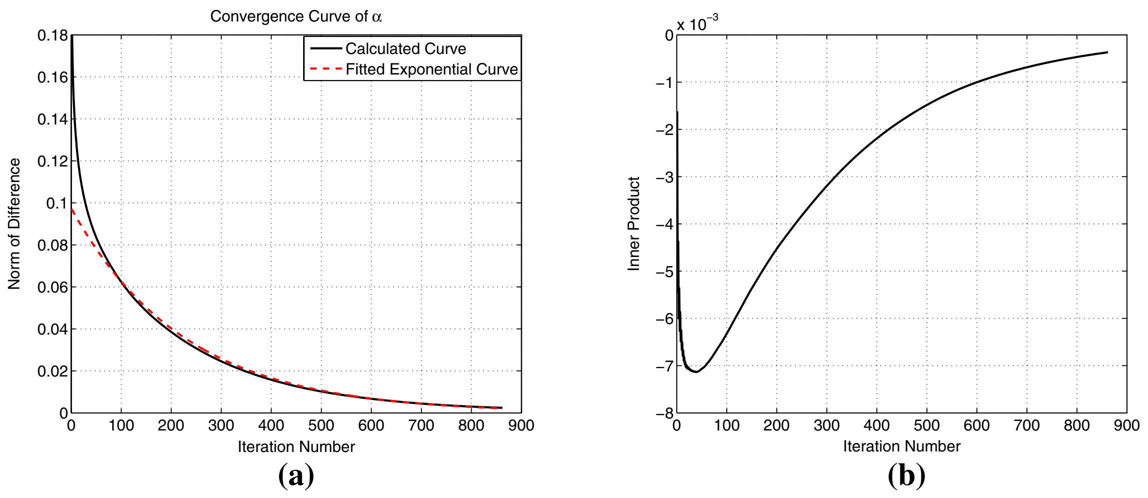

Fig. 2 **a** On the CBCL face training dataset, the convergence of the L-SVDD solution $\boldsymbol{\alpha}^k$ to the QP-SVDD solution $\boldsymbol{\alpha}_{\mathrm{Q}}$ in terms of the norm of the difference vector. **b** The evolution of the inner product $(\boldsymbol{\alpha}^k)'(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v})$ with the iteration number

Table 1 The training time (t, in seconds) of QP-SVDD with different training set size (n)

Training set size (n)	400	800	1200	1600	2000	2400
Training time (t)	21.8	218.6	850.1	2090.7	4033.0	6864.7
$t/n^3(\times10^{-6})$	0.3405	0.4270	0.4920	0.5104	0.5041	0.4966

For the purpose of complexity analysis, the values for $t/n^3(\times10^{-6})$ are also listed

than QP-SVDD, while still possessing similar classification performance.

QP-SVDD found 79 support vectors, while L-SVDD extracted 85 support vectors, with the precision rate 92.94% and the recall rate 100%. These numbers indicate that L-SVDD and QP-SVDD obtain models with similar complexity. The precision and recall rates show that L-SVDD correctly finds all the support vectors while only mistakenly regards a few training examples as support vectors. We should mention that, by mathematical and experimental analysis, the original SVDD papers [31, 33] conclude that, for the SVDD model with Gaussian kernel, the number of support vectors decreases as the kernel parameter σ or the control parameter C increases. This conclusion should be true for L-SVDD because it is just another solution (although approximate) to the same problem. We tested this conjecture by changing different parameter settings, and found that the number of support vectors for L-SVDD also follows the conclusion presented in [31, 33]. Since this issue is not particular to L-SVDD, we choose not to present the results.

We denote the solution of QP-SVDD as $\boldsymbol{\alpha}_{\mathrm{Q}}$, and calculate the norm of the difference between the k-th iteration solution of L-SVDD and the QP-SVDD solution, that is, $\|\boldsymbol{\alpha}^k - \boldsymbol{\alpha}_{\mathrm{Q}}\|$. The black curve in Fig. 2a represents the evolution of the norm $\|\boldsymbol{\alpha}^k - \boldsymbol{\alpha}_{\mathrm{Q}}\|$, which shows that the difference indeed decreases to zero exponentially. We further fit the obtained values of $\|\boldsymbol{\alpha}^k - \boldsymbol{\alpha}_{\mathrm{Q}}\|$ to an exponential function Ac^k, and we plot the fitted curve in Fig. 2a as the red dashed curve. We see that the calculated values and the fitted values are close to each other, and this numerically verifies our theoretical result in Eq. (23).

The purpose of Algorithm 1 is to solve Eq. (14) with respect to $\boldsymbol{\alpha}$. To observe the evolution of the solution, Fig. 2b plots the inner product $(\boldsymbol{\alpha}^k)'(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v})$, which approaches 0 as the training program proceeds. In fact, when the algorithm terminated, the inner product value was -3.6883×10^{-4}, very close to 0.

To numerically investigate the relationship between the training time of QP-SVDD and the training set size, we gradually increase the training set size and record the training time (in seconds) of QP-SVDD, and the results are given in Table 1, in which we also list the ratio between the training time and the cubic of training set size. Table 1 shows that, with the increasing of training set size, the training time increases dramatically, and the majority of the ratio t/n^3 are around 0.4618×10^{-6} (average of the 3rd row). This suggests that the training time complexity of QP-SVDD is around $O(n^3)$.

We conducted the same experiment with L-SVDD, and Table 2 gives the training time of L-SVDD for different training set size, and the number of iterations needed are

Table 2 The training time (t, in seconds) of L-SVDD with different training set size (n)

Training set size (n)	400	800	1200	1600	2000	2400
Training time (t)	0.1092	0.5772	1.5912	3.5256	7.0512	11.8249
# Iteration	191	347	441	560	726	853
Time per iteration (tpi)	0.0006	0.0017	0.0036	0.0063	0.0097	0.0139
$tpi/n^2(\times 10^{-8})$	0.3573	0.2599	0.2506	0.2459	0.2428	0.2407

For the purpose of time complexity analysis, the iteration numbers needed to converge, the time per iteration (tpi), and tpi/n^2 are also listed

Table 3 On a subset from face data, the values of $\sum_{i=1}^{n} \alpha_i$ with different parameter settings

ρ	100	150	200	300	400	500	600
$C = 1$	0.9982	0.9988	0.9991	0.9994	0.9996	0.9996	0.9997
$C = 2$	0.9983	0.9988	0.9991	0.9994	0.9996	0.9997	0.9997
$C = 4$	0.9983	0.9989	0.9991	0.9994	0.9996	0.9997	0.9997
$C = 8$	0.9983	0.9989	0.9991	0.9994	0.9996	0.9997	0.9997

Table 4 On a subset of face data, the number of iterations needed for L-SVDD to converge with different values of γ

γ	0.9 / C	0.92 / C	0.94 / C	0.95 / C	0.97 / C	0.98 / C	0.99 / C
# Iteration	516	508	499	496	487	483	479

Table 5 On a subset of face data, the number of iterations needed for L-SVDD to converge with different values of C

C	1	2	4	6	8	10	16
# Iteration	145	272	495	685	852	1012	1425

Table 6 On a subset of face data, the number of iterations needed for L-SVDD to converge with different values of ρ

ρ	100	150	200	300	400	500
# Iteration	495	495	496	494	492	489

also shown. To verify our theoretical analysis in Remark 2, we calculated the time per iteration (tpi), and the ratio between tpi and n^2, which are also presented in Table 2. Table 2 shows that the values of tpi/n^2 are around a constant, i.e., 0.2662×10^{-8} (average of the 5th row). This indicates that the time per iteration is of order $O(n^2)$, hence the total training time complexity of L-SVDD is about the iteration number multiplying $O(n^2)$, which is consistent with the theoretical analysis in Remark 2.

In the derivation of Eq. (11), to absorb the constraint $\sum_{i=1}^{n} \alpha_i = 1$ in Eq. (10), we introduced a parameter ρ in the penalty function, and the penalty function method requires that ρ should be large enough to ensure the validity of the constraint [28, Sect. 6.2.2]. To verify this, we randomly selected 600 face images as training data and run the L-SVDD algorithm for different settings of ρ and C (since γ depends on C through $\gamma = 0.95/C$). Table 3 gives the values of $\sum_{i=1}^{n} \alpha_i$ for different parameter settings. The results show that as long as ρ large enough, $\sum_{i=1}^{n} \alpha_i$ is reasonably close to 1. Moreover, for a fixed ρ, $\sum_{i=1}^{n} \alpha_i$ is almost the same for a variety of C values. This paper sets $\rho = 200$, which is sufficient to make the constraint hold approximately.

To experimentally investigate the impact of parameters C, ρ, and γ to the convergence behavior of L-SVDD, we randomly select 600 face images, and run L-SVDD training algorithm with different parameter settings. We first fix $C = 4$ and $\rho = 200$, and change the value of γ. Table 4 lists the number of iterations to converge along with different values of γ. It is observed that as γ increases, the algorithm converges faster. Next, we fix $\rho = 100$, set $\gamma = 0.95/C$, and change the value of C, and Table 5 gives the number of iterations to converge along with different values of

Table 7 The iteration numbers needed and the training time (in seconds) of IPM-SVDD and L-SVDD with different parameter settings (γ was set to be 0.95 / C)

λ:	100			200		
C :	2	4	8	2	4	8
# Iter IPM-SVDD	28	29	29	28	29	29
Time IPM-SVDD	37.2530	37.6430	38.0330	36.6758	37.6742	37.7366
# Iter L-SVDD	272	495	852	272	496	846
Time L-SVDD	0.2808	0.4836	0.6864	0.3432	0.4368	0.7176

C. We see that a smaller C will result in faster convergence. Finally, we fix $C = 4$ and $\gamma = 0.95/C$, and change the parameter ρ. Table 6 shows the number of iterations to converge with different values of ρ. It is evident that ρ hardly has any significant influence to the convergence rate. All these numerical results are consistent with our theoretical analysis in Sect. 3.2.

To compare the interior point method based SVDD (IPM-SVDD) to the proposed L-SVDD, we randomly select 600 face images and train the two SVDD models with different parameter settings, and the convergence measures are given in Table 7. Table 7 shows that, with all the parameter settings, although IPM-SVDD converges in fewer iterations, it spends considerably more training time than L-SVDD (i.e., 50–100 times slower). This is expected due to the reason that was stated in Sect. 3.3. We also notice that in all cases, the difference between the IPM solution ($\boldsymbol{\alpha}_I$) and L-SVDD solution ($\boldsymbol{\alpha}_L$) was within 10^{-4} in terms of the norm of the difference vector, i.e., $\|\boldsymbol{\alpha}_I - \boldsymbol{\alpha}_L\| < 10^{-4}$, and this means that the two methods get very close models.

In our experiments, it was also observed that if we set the training set size $n = 1200$, the IPM-SVDD training program would have memory issue on our computer but L-SVDD did not have such problem even for $n = 2400$. This is because IPM needs variables for gradient, Hessian, search directions, and other intermediate results, while L-SVDD does not have any intermediate result to store in memory. From our analysis and experimental results, we may claim that the proposed L-SVDD is not only time efficient but also space efficient, compared to IPM-SVDD.

4.3 Handwritten digit recognition

The handwritten digits dataset consists of 7291 training examples and 2007 testing examples, and is available at https://web.stanford.edu/~hastie/ElemStatLearn/. The dataset consists of normalized handwritten digits (“0” to “9”), each is a 16×16 gray-scale image. Same as the experiment in Sect. 4.2, we simply use these 256 pixel values as inputs to the algorithms.

To compare the performance of L-SVDD to that of QP-SVDD, we created ten binary classification problems, with each digit as positive class, respectively. The performance measures of QP-SVDD and L-SVDD are given in Table 8, which lists the number of support vectors, AUC on testing set, and training time for each algorithm. We observe that for all the ten problems, the AUC of both algorithms are quite close, since the difference between AUC is at most 0.001, and most problems have AUC difference even under 0.0004. The ROC curves of QP-SVDD and L-SVDD, similar to the results on CBCL face data, are almost indistinguishable (also indicated by the AUC values), we thus choose not to present the ROC curves.

Table 8 also lists the training times of QP-SVDD and L-SVDD on all the ten problems, and the sizes of the considered problems are also given. It is observed that for the problems with size under 1000, L-SVDD terminates within 1 second; for the two problems with larger size, L-SVDD needs 2–3 s. To clearly illustrate the speed advantage of L-SVDD, Table 8 lists the ratio between the training times of QP-SVDD and L-SVDD for different problems. It is clear that on most of the problems, L-SVDD is 200–400 times faster than the QP counterpart. More importantly, we should mention that in our implementation, the core quadratic programming code for QP-SVDD was developed in C++ which is much more computationally efficient than MATLAB, in which L-SVDD was developed. Taking this factor into account, L-SVDD would be much more time-efficient than QP-SVDD, if they were implemented in the same programming language and ran on the same platform.

To gain further insight about the computational complexity of QP-SVDD, we compare the training time of QP-SVDD on each problem. Since the problem for digit “8” has the smallest size, we use it as the baseline. We calculate the ratio of the training time on each digit to that on digit “8”, along with the cubic of the problem size ratio, and the results are given in Table 9. Table 9 shows that the two numbers are close enough for each problem, which indicates that the training time of QP-SVDD grows roughly in the rate of $O(n^3)$.

Table 11 lists the number of iterations needed for L-SVDD training algorithm to converge. We see that the algorithm usually terminates in 600 iterations, with only one exception. To analyze the computational complexity of L-SVDD, we first calculate the average time spent on one iteration for each problem, using the information given in Tables 8 and 11. We denote the results as tpi_0 through tpi_9 (“tpi” stands for “time per iteration”). Same as the analysis

Table 8 On the handwritten digit dataset, the performance comparison between QP-SVDD and L-SVDD

Digit	# Train	Method	# SV	AUC	Training time	Time ratio
"0"	1194	QP-SVDD	109	0.9811	867.8648	409.0615
		L-SVDD	111	0.9813	2.1216	
"1"	1005	QP-SVDD	18	0.9905	478.6579	158.9803
		L-SVDD	19	0.9905	3.0108	
"2"	731	QP-SVDD	128	0.8977	165.7199	366.3128
		L-SVDD	128	0.8973	0.4524	
"3"	658	QP-SVDD	82	0.9480	121.6184	236.2440
		L-SVDD	84	0.9485	0.5148	
"4"	652	QP-SVDD	87	0.9396	117.4064	268.7875
		L-SVDD	88	0.9405	0.4368	
"5"	556	QP-SVDD	94	0.8889	70.6685	283.1270
		L-SVDD	95	0.8891	0.2496	
"6"	664	QP-SVDD	81	0.9785	124.8008	285.7161
		L-SVDD	82	0.9783	0.4056	
"7"	645	QP-SVDD	62	0.9711	113.3035	258.0662
		L-SVDD	63	0.9709	0.4836	
"8"	542	QP-SVDD	86	0.9165	66.8152	237.9459
		L-SVDD	87	0.9168	0.2808	
"9"	644	QP-SVDD	65	0.9752	114.5671	253.2429
		L-SVDD	66	0.9753	0.4524	

The listed are the training set size, the number of support vectors found by each method, the AUC on the testing set by each method, and the training times in seconds. The last column gives the ratio between the training times of the two algorithms on each problem

Table 9 For QP-SVDD, the cubic of problem size ratios and the training time ratios on the handwritten digit dataset

$(n_0/n_8)^3 = 10.69$	$(n_1/n_8)^3 = 6.38$	$(n_2/n_8)^3 = 2.45$	$(n_3/n_8)^3 = 1.79$	$(n_4/n_8)^3 = 1.74$
$t_0/t_8 = 12.98$	$t_1/t_8 = 7.16$	$t_2/t_8 = 2.48$	$t_3/t_8 = 1.82$	$t_4/t_8 = 1.76$
$(n_5/n_8)^3 = 1.08$	$(n_6/n_8)^3 = 1.84$	$(n_7/n_8)^3 = 1.68$	$(n_8/n_8)^3 = 1$	$(n_9/n_8)^3 = 1.67$
$t_5/t_8 = 1.06$	$t_6/t_8 = 1.87$	$t_7/t_8 = 1.69$	$t_8/t_8 = 1$	$t_9/t_8 = 1.71$

The results show that QP-SVDD roughly has computational complexity $O(n^3)$

Table 10 For L-SVDD, the square of problem size ratios and the ratio of time per iteration in training on the handwritten digit dataset

$(n_0/n_8)^2 = 4.85$	$(n_1/n_8)^2 = 3.44$	$(n_2/n_8)^2 = 1.82$	$(n_3/n_8)^2 = 1.47$	$(n_4/n_8)^2 = 1.45$
$tpi_0/tpi_8 = 4.54$	$tpi_1/tpi_8 = 2.76$	$tpi_2/tpi_8 = 2.00$	$tpi_3/tpi_8 = 1.43$	$tpi_4/tpi_8 = 1.58$
$(n_5/n_8)^2 = 1.05$	$(n_6/n_8)^2 = 1.50$	$(n_7/n_8)^2 = 1.42$	$(n_9/n_8)^2 = 1$	$(n_9/n_8)^2 = 1.41$
$tpi_5/tpi_8 = 0.99$	$tpi_6/tpi_8 = 1.39$	$tpi_7/tpi_8 = 1.43$	$tpi_8/tpi_8 = 1$	$tpi_9/tpi_8 = 1.43$

The results show that L-SVDD roughly has computational complexity $O(n^2)$ multiplying the iteration number

for QP-SVDD, we use digit "8" as the baseline, and calculate the ratio of the time per iteration in training on each digit to that on digit "8", along with the square of the problem size ratio. The results are presented in Table 10, which shows that these numbers are reasonably close for each problem, and this indicates that the time per iteration grows roughly with order $O(n^2)$. Consequently, the time complexity of training L-SVDD (i.e., Algorithm 1) is about $O(n^2)$ multiplying the iteration number, which is consistent with the result in Sect. 4.2. These analysis verifies our theoretical analysis in Remark 2.

In our implementations for QP-SVDD and L-SVDD, the kernel matrix was calculated beforehand, thus the time spent on calculating kernel matrix was not counted as training time. In fact, computing kernel matrix spent much more time than Algorithm 1 itself in our experiments.[6] Our theoretical

[6] For example, for digit "0", our computer spent 47.0655 s on constructing kernel matrix, while only 2.1216 s on Algorithm 1.

Table 11 Comparison of the support vectors extracted by QP-SVDD and those extracted by L-SVDD, in terms of precision and recall rates

Digit	"0"	"1"	"2"	"3"	"4"
Precision	98.20%	94.74%	100%	97.62%	98.86%
Recall	100%	100%	100%	100%	100%
#Iteration	604	1410	292	465	357
Digit	"5"	"6"	"7"	"8"	"9"
Precision	98.95%	98.78%	98.41%	98.85%	97.01%
Recall	100%	100%	100%	100%	100%
#Iteration	325	378	438	363	409

This table also presents the iteration numbers needed for L-SVDD to converge

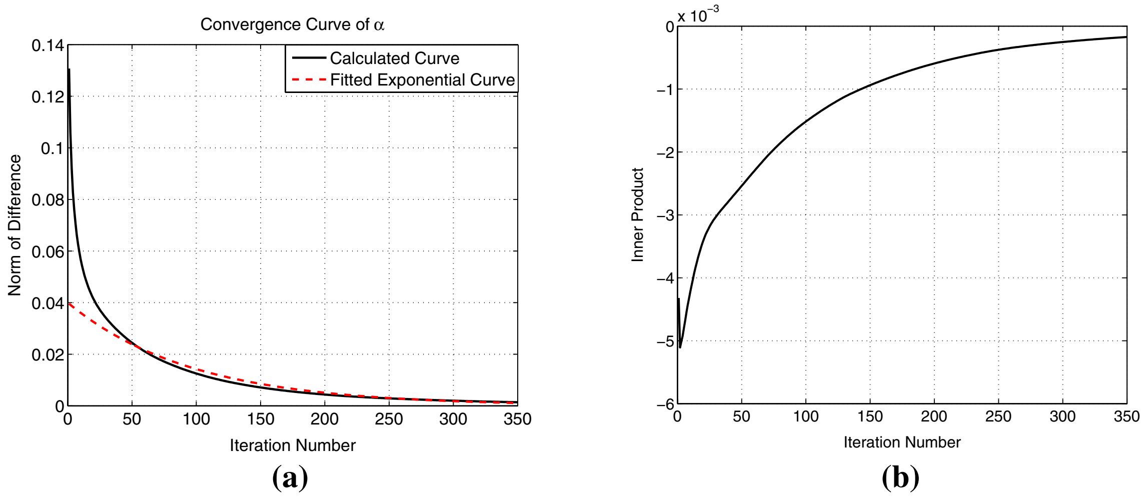

Fig. 3 **a** On digit "4", the convergence of the L-SVDD solution $\boldsymbol{\alpha}^k$ to the QP-SVDD solution $\boldsymbol{\alpha}_Q$ in terms of the norm of the difference vector. **b** On digit "4", the evolution of the inner product $(\boldsymbol{\alpha}^k)'(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v})$ with the iteration number

analysis of the computational complexity for QP-SVDD and L-SVDD is consistent with the experimental results, but the order of complexity seems not to support the results presented in Table 8, which shows that L-SVDD is much more time-efficient than QP-SVDD.

To explain this apparent paradox, we write the training time of QP-SVDD as $t_Q = k_1 n^3$, and that of L-SVDD could be written as $t_L = k_2 M n^2$, where M is the iteration number of L-SVDD, n is the training set size, and k_1 and k_2 are constants. Using the information form Table 8, we can calculate that in average $k_1 \approx 4.3644 \times 10^{-7}$; and we calculate $k_2 \approx 2.5785 \times 10^{-9}$ from Tables 8 and 11. Note that these numbers are comparable to those obtained in Sect. 4.2 from Tables 1 and 2. Therefore, k_1 is much larger than k_2 in average (about 170 times), and this explains why L-SVDD is so much more time-efficient than the QP counterpart.

Table 8 shows that QP-SVDD and L-SVDD extract roughly the same number of support vectors. To investigate the overlap of the sets of support vectors, we calculate the precision and recall rates, given in Table 11. The results show that all the recall rates are 100%, which means that L-SVDD extracts all the support vectors which are found by QP-SVDD, and the very high precision rates demonstrate that L-SVDD only mistakenly regards a few training examples as support vectors.

Similar to the experiment on face dataset, on the problem for digit "4", we calculate the norm of difference between the vector $\boldsymbol{\alpha}^k$ from the k-th iteration of L-SVDD and the vector $\boldsymbol{\alpha}_Q$ from QP-SVDD, shown as the black curve in Fig. 3a, which also gives the fitted exponential function as the red dashed curve. We once again see that the calculated values and the fitted values are very close to each other, and this numerically verifies our theoretical analysis presented in Eq. (23). Figure 3b presents the inner product $(\boldsymbol{\alpha}^k)'(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{v})$ with the iteration number on the problem of digit "4", which clearly shows that the inner product approaches 0 as the training algorithm proceeds. In fact, when the algorithm terminated, the inner product value was -1.6360×10^{-4}. The curves on other digits give the same pattern, thus we do not show them all.

5 Conclusion and future works

Support vector data description (SVDD) is a well known tool for data description and patter classification, with wide applications. In literature, the SVDD model is often trained by solving the dual of a constrained optimization problem, resulting in a quadratic programming. However, the quadratic programming is computationally expensive to solve, with time complexity about $O(n^3)$, where n is the training set size.

Using the sum of the squared error and the idea of quadratic penalty function method, we formulate the Lagrangian dual problem of SVDD as minimizing a convex quadratic function over the positive orthant, which can be solved efficiently by a simple iterative algorithm. The proposed Lagrangian SVDD (L-SVDD) algorithm is very easy to implement, requiring no particular optimization toolbox other than basic matrix operations. Theoretical and experimental analysis show that the L-SVDD solution converges to the Quadratic Programming based SVDD (QP-SVDD) solution r-linearly.

Extensive experiments were conducted on various pattern classification problems, and we compared the performance of L-SVDD to that of QP-SVDD, in terms of ROC curve measures and the training time. Our results show that L-SVDD has similar classification performance as its QP counterpart; both L-SVDD and QP-SVDD extract almost the same set of support vectors. However, L-SVDD is a couple hundreds times faster than QP-SVDD in training. The experiments also verified the theoretical analysis about the convergence rate and the training time complexity of L-SVDD.

Due to the limit of available computing resource, we did not test L-SVDD on larger training set. However, if the training set is too large, we conjecture that the performance of L-SVDD might deteriorate. One reason is that in Algorithm 1 for L-SVDD, we need to calculate the inverse of an $n \times n$ matrix $\mathbf{Q}$, where n is the training set size. It is well known that inverting a large matrix is numerically unreliable and time/memory consuming. Another reason is that Algorithm 1 involves matrix and vector multiplication, whose computing cost scales up with n^2. This work verifies the speed advantage of L-SVDD when the training set size is at order of several thousands. It would be insightful to investigate the performance of L-SVDD for larger scaled problems.

This paper formulates the optimization problem for L-SVDD as a linear complementarity problem shown in Eq. (14), and we propose Algorithm 1 to solve it. In optimization literature, there are many methods to solve linear complementarity problem, for example, Newton's method [1], pivoting method [16], and the methods in [10]. Thus, as next step of work, we plan to apply these methods to L-SVDD model.

Similar to the works on support vector machine for classification [12] and regression [3], we can remove the constraint in Eq. (13) by adding extra penalty terms, obtaining an unconstrained optimization problem for an approximated SVDD model. Then gradient based optimization methods can be applied to this approximation, for example, Newton's method [5], coordinate gradient descent [6], block-wise coordinate descent [19], or conjugate gradient method [39, 40]. This is another possible extension to the current work.

Acknowledgements The author would like to thank the editors and four anonymous reviewers for their constructive suggestions which greatly helped improve the paper. This work was supported by a Summer Faculty Fellowship from Missouri State University.

Orthogonality condition for two nonnegative vectors

We show that two nonnegative vectors $\mathbf{a}$ and $\mathbf{b}$ are perpendicular, if and only if $\mathbf{a} = (\mathbf{a} - \gamma\mathbf{b})_+$ for any real $\gamma > 0$.

If two nonnegative real numbers a and b satisfy $ab = 0$, there is at least one of a and b is 0. If $a = 0$ and $b \geq 0$, then for any $\gamma > 0$, $a - \gamma b \leq 0$, so that $(a - \gamma b)_+ = 0 = a$; if $a > 0$, we must have $b = 0$, then for any real $\gamma > 0$, $(a - \gamma b)_+ = (a)_+ = a$. In both cases, there is $a = (a - \gamma b)_+$ for any real number $\gamma > 0$.

Conversely, assume that two nonnegative real numbers a and b can be written as $a = (a - \gamma b)_+$ for any real number $\gamma > 0$. If a and b are both strictly positive, then $a - \gamma b < a$ since $\gamma > 0$. Consequently, $(a - \gamma b)_+ < a$, which is contradict to the assumption that $a = (a - \gamma b)_+$. Thus at least one of a and b must be 0, i.e., $ab = 0$.

Now assume that nonnegative vectors $\mathbf{a}$ and $\mathbf{b}$ in space $\mathbb{R}^p$ are perpendicular, that is, $\sum_{i=1}^{p} a_i b_i = 0$. Since both of a_i and b_i are nonnegative, there must be $a_i b_i = 0$ for $i = 1, 2, \ldots, p$. By the last argument, this is equivalent to $a_i = (a_i - \gamma b_i)_+$ for any $\gamma > 0$ and any $i = 1, 2, \ldots, p$. In vector form, we have $\mathbf{a} = (\mathbf{a} - \gamma\mathbf{b})_+$.

References

1. Aganagić M (1984) Newton's method for linear complementarity problems. Math Program 28(3):349–362
2. Armand P, Gilbert JC, Jan-Jégou S (2000) A feasible BFGS interior point algorithm for solving convex minimization problems. SIAM J Optim 11(1):199–222

3. Balasundaram S, Gupta D, Kapil (2014) Lagrangian support vector regression via unconstrained convex minimization. Neural Netw 51:67–79
4. Ben-Hur A, Horn D, Siegelmann HT, Vapnik V (2001) Support vector clustering. J Mach Learn Res 2:125–137
5. Bertsekas DP (1999) Nonlinear programming, 2nd edn. Athena Scientific, Belmont, MA
6. Chang K-W, Hsieh C-J, Lin C-J (2008) Coordinate descent method for large-scale L_2-loss linear support vector machines. J. Mach Learn Res 9:1369–1398
7. Chang CC, Lin CJ (2011) LIBSVM: a library for support vector machines. ACM Trans Intell Syst Technol 2(3):27:1–27:27
8. Chapelle O, Vapnik V, Bousquet O, Mukherjee S (2002) Choosing multiple parameters for support vector machines. Mach Learn 46(1–3):131–159
9. Cottle RW (1983) On the uniqueness of solutions to linear complementarity problems. Math Program 27(2):191–213
10. Cottle RW, Pang J-S, Stone RE (1992) The linear complementarity problem. SIAM, Philadelphia, PA
11. Fawcett T (2006) An introduction to ROC analysis. Pattern Recognit Lett 27(8):861–874
12. Fung G, Mangasarian OL (2003) Finite Newton method for Lagrangian support vector machine classification. Neurocomputing 55:39–55
13. Gold C, Sollich P (2003) Model selection for support vector machine classification. Neurocomputing 55(1–2):221–249
14. Gunn SR (1997) Support vector machines for classification and regression, technical report, image speech and intelligent systems research group, University of Southampton. http://users.ecs.soton.ac.uk/srg/publications/pdf/SVM.pdf
15. Joachims J (1999) Making large-scale SVM learning practical. In: chölkopf B, Burges SC, Smola A (eds)Advances in Kernel methods—support vector learning. MIT-Press
16. Kremers H, Talman D (1994) A new pivoting algorithm for the linear complementarity problem allowing for an arbitrary starting point. Math Program 63(1):235–252
17. Lee D, Lee J (2007) Domain described support vector classifier for multi-classification problems. Pattern Recogn 40(1):41–51
18. Lee S-W, Park J, Lee S-W (2006) Low resolution face recognition based on support vector data description. Pattern Recogn 39(9):1809–1812
19. Liu H, Palatucci M, Zhang J (2009) Blockwise coordinate descent procedures for the multi-task Lasso, with applications to neural semantic basis discovery. In: Proceedings of the 26th Int'l Conf. on Machine Learning, pp 649–656
20. Mangasarian OL (1994) Nonlinear programming. SIAM, Philadelphia, PA
21. Mangasarian OL, Musicant DR (2001) Lagrangian support vector machines. J Mach Learn Rese 1:161–177
22. Muñoz-Marí J, Bruzzone L, Camps-Valls G (2007) A support vector domain description approach to supervised classification of remote sensing images. IEEE Trans Geosci Remote Sens 45(8):2683–2692
23. Musicant DR, Feinberg A (2004) Active set support vector regression. IEEE Trans Neural Netw 15(2):268–275
24. Osuna E, Freund R, Girosi F (1997) An improved training algorithm for support vector machines. In: Proceedings of IEEE Workshop Neural Networks for Signal Processing, pp 276–285
25. Osuna E, Freund R, Girosi F (1997) Training support vector machines: an application to face detection. In: Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition
26. Platt J (1998) Fast training of support vector machines using sequential minimal optimization. In: Schölkopf B, Burges C, Smola A (eds) Advances in Kernel methods—support vector learning. MIT-Press
27. Powers DMW (2011) Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. J Mach Learn Technol 2(1):37–63
28. Ruszczyński A (2006) Nonlinear optimization. Princeton University Press, Princeton, NJ
29. Tang R, Han J, Zhang X (2009) Efficient iris segmentation method with support vector domain description. Opt Appl XXXIX(2):365–374
30. Tax DMJ, Duin RPW (1999) Data domain description using support vectors. Neural Networks, Proc. of the European Symposium on Artificial, pp 251–256
31. Tax DMJ, Duin RPW (1999) Support vector domain description. Pattern Recogn Lett 20(11–13):1191–1199
32. Tax DMJ, Duin RPW (2002) Uniform object generation for optimizing one-class classifiers. J Mach Learn Res 2:155–173
33. Tax DMJ, Duin RPW (2004) Support vector data description. Mach Learn 54(1):45–66
34. Vapnik V (1998) Statistical learning theory. Wiley, NY
35. Viola P, Jones M (2001) Rapid object detection using a boosted cascade of simple features. In: Proc. of IEEE Conf. on Computer Vision and Pattern Recognition
36. Wahba G, Lin Y, Zhang H (2000) Generalized approximate cross validation for support vector machines, or, another way to look at margin-like quantities. In: Smola B, Scholkopf, Schurmans (eds) Advances in large margin classifiers. MIT Press
37. Wang X, Lu S, Zhai J (2008) Fast fuzzy multicategory SVM based on support vector domain description. Int J Patt Recogn Artif Intell 1:109–120
38. Yu X, Dementhon D, Doermann D (2008) Support vector data description for image categorization from internet images. In: Proc. of IEEE International Conf. on Pattern Recognition
39. Zheng S (2015) A fast algorithm for training support vector regression via smoothed primal function minimization. Int J Mach Learn Cybernet 6(1):155–166
40. Zheng S (2016) Smoothly approximated support vector domain description. Pattern Recogn 49(1):55–64