Advances in Computational Intelligence (2022) 2:9
https://doi.org/10.1007/s43674-021-00026-4

ORIGINAL ARTICLE

A support vector approach based on penalty function method

Songfeng Zheng[1]

Received: 24 March 2021 / Revised: 4 October 2021 / Accepted: 12 October 2021 / Published online: 17 December 2021

Abstract
Support vector machine (SVM) models are usually trained by solving the dual of a quadratic programming, which is time consuming. Using the idea of penalty function method from optimization theory, this paper combines the objective function and the constraints in the dual, obtaining an unconstrained optimization problem, which could be solved by a generalized Newton method, yielding an approximate solution to the original model. Extensive experiments on pattern classification were conducted, and compared to the quadratic programming-based models, the proposed approach is much more computationally efficient (tens to hundreds of times faster) and yields similar performance in terms of receiver operating characteristic curve. Furthermore, the proposed method and quadratic programming-based models extract almost the same set of support vectors.

Keywords Support vectors · penalty function · quadratic programming · Newton algorithm.

1 Introduction

For more than two decades, support vector machine (SVM) classifier (Cortes and Vapnik 1995) and support vector data description (SVDD) (Tax and Duin 1999, 2004) have attracted much attention in research and have been successfully applied to various scenarios. In the training phase, the formulations of SVM and SVDD lead to a quadratic programming (Cortes and Vapnik 1995; Tax and Duin 1999, 2004). Although the decomposition techniques (Osuna et al. 1997a, b) or sequential minimization methods (Platt 1998) could be employed to solve the quadratic programming, the training of SVM/SVDD has time complexity about $O(n^3)$, where n is the training set size. Therefore, training an SVM/SVDD model is time consuming, especially for large training set. As such, given their wide applications, it is highly desirable to develop a time-efficient yet accurate enough training algorithm for SVM/SVDD. Furthermore, since the support vectors contain important information for SVM/SVDD models, we also want to obtain the support vector information from the fast algorithm.

Instead of relying on a special quadratic programming solver for SVM and SVDD, we apply the idea of quadratic penalty function method from optimization literature (Ruszczyński 2006), converting the constrained quadratic programming to an unconstrained optimization problem. Then, a generalized Newton method, which is known to converge fast, is applied to solve the obtained problem, such that an approximated SVM or SVDD model could be obtained. The proposed algorithms for SVM and SVDD are easy to implement, without requiring particular optimization toolbox other than a standard linear system solver.

We tested the proposed algorithms on several pattern classification problems, and detailed performance comparison demonstrates that the proposed Newton algorithm-based SVM (N-SVM) and SVDD (N-SVDD) often yield similar performances to those of the quadratic programming based SVM (QP-SVM) and SVDD (QP-SVDD), in terms of area under the receiver operating characteristic (ROC) curve and the support vectors extracted. However, N-SVM and N-SVDD are much more computationally efficient (often tens to hundreds of times faster) in training than their quadratic programming-based counterparts.

In literature, to avoid the expensive quadratic programming in training SVM-type models, gradient-based optimization methods were considered. For example, in Lee and Mangasarian (2001) and Zheng (2016), a smooth approximation of the loss functions for SVM and SVDD was applied so that gradient descent algorithm could be applied to the approximated primal objective function, resulting computationally efficient algorithms. Newton's method was applied to minimize the primal objective function in SVM with L_2 and

✉ Songfeng Zheng
SongfengZheng@MissouriState.edu

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

Huber loss function in Chapelle (2007). Stochastic gradient descent was directly used in Shalev-Schwartz et al. (2011) and Wang et al. (2012). However, these methods directly optimize the primal objective function (or its modified versions) of SVM or SVDD, hence could not find the support vectors. On the contrary, the proposed idea works on the dual problem over the Lagrangian multipliers $\boldsymbol{\alpha}$, so that the support vectors could be identified, which are important to save predicting time for nonlinear classifiers and estimating the generalization error of SVM (Opper and Winther 2000; Vapnik and Chapelle 2000).

The rest of this paper is organized as follows: Sect. 1.1 introduces the notations and mathematical tools used in this paper; Sect. 2 briefly reviews the formulations of SVM and SVDD models; Sect. 3 applies the quadratic penalty function method to formulate the quadratic programming as an unconstrained optimization problem, and a generalized Newton algorithm is introduced to solve the obtained problem; Sect. 4 compares the performance measures in terms of ROC curve analysis and training time of the proposed Newton algorithms to those of QP-SVM/QP-SVDD on four real-world datasets, and we also compare the support vectors extracted by the two methods; Sect. 5 summarizes this paper and discusses some future research directions.

1.1 Notations

All scalars are represented by lower case symbols. 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 vectors $\mathbf{a}$ and $\mathbf{b}$ in $\mathbb{R}^n$, $\mathbf{a} \geq \mathbf{b}$ means $a_i \geq b_i$ for each $i = 1, \ldots, n$. For vector $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|$ stands for the 2-norm of $\mathbf{x}$, that is, $\|\mathbf{x}\| = \sqrt{x_1^2 + \cdots + x_n^2}$. The plus function $\mathbf{x}_+$ is defined as $(\mathbf{x}_+)_i = \max\{0, x_i\}$, for $i = 1, \ldots, n$. The subgradient of $\mathbf{x}_+$ is denoted by $\mathbf{x}_*$, which is a step function defined as $(\mathbf{x}_*)_i = 1$ if $x_i > 0$, $(\mathbf{x}_*)_i = 0$ if $x_i < 0$, and $(\mathbf{x}_*)_i \in [0, 1]$ if $x_i = 0$, for $i = 1, \ldots, n$. If $x_i = 0$, we typically take $(\mathbf{x}_*)_i = 0.5$. A column vector of ones (zeros) in $\mathbb{R}^n$ will be denoted by $\mathbf{1}_n$ ($\mathbf{0}_n$), and the identity matrix of n-th order will be denoted by $\mathbf{I}_n$.

If f is a real-valued function defined on $\mathbb{R}^n$, the gradient of f at $\mathbf{x}$ is denoted by $\nabla f(\mathbf{x})$ which is a column vector in $\mathbb{R}^n$, and the Hessian of f at $\mathbf{x}$ is denoted by $\nabla^2 f(\mathbf{x})$, which is an $n \times n$ matrix. For a piecewise quadratic function $f(\mathbf{x}) = \frac{1}{2}\|(\mathbf{A}\mathbf{x} - \mathbf{b})_+\|^2$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$, the gradient vector is $\nabla f(\mathbf{x}) = \mathbf{A}'(\mathbf{A}\mathbf{x} - \mathbf{b})_+$, which is not differentiable, thus the ordinary Hessian of f does not exist. However, we can define its generalized Hessian (Hiriart-Urruty et al. 1984) which is the $n \times n$ symmetric positive semi-definite matrix

$$\partial^2 f(\mathbf{x}) = \mathbf{A}' \mathrm{diag}(\mathbf{A}\mathbf{x} - \mathbf{b})_* \mathbf{A},$$

where $\mathrm{diag}(\mathbf{A}\mathbf{x} - \mathbf{b})_*$ denotes an $m \times m$ diagonal matrix with diagonal elements $(\mathbf{A}_i \mathbf{x} - b_i)_*$, for $i = 1, \ldots, m$, with $\mathbf{A}_i$ being the i-th row of matrix $\mathbf{A}$.

2 Support vector machine and support vector data description

In this section, we briefly review the formulations of support vector machine and support vector data description.

2.1 Support vector machine

For two-class classification problem, assume that the given training dataset is $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ with $\mathbf{x}_i \in \mathbb{R}^p$ as the feature vector and $y_i \in \{-1, 1\}$ as the class label. The idea of support vector machine (SVM) is to first map the feature vector into some high-dimensional reproducing kernel Hilbert space $\mathcal{H}$ by a mapping $\phi(\mathbf{x})$, and then construct a linear classifier in $\mathcal{H}$ with the form $\mathbf{w}'\phi(\mathbf{x}) + b$. SVM is constructed so that the margin of the classifier is large, which is measured by $1/\|\mathbf{w}\|$. To allow for possible mistakes, we also introduce a set of slack variables $\xi_i \geq 0$, which represents the penalty to the classifier for making a mistake at $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, n$.

The SVM model can be fitted by solving the following optimization problem

$$\begin{cases} \min_{\mathbf{w}, \boldsymbol{\xi}, b} & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^n \xi_i \\ \text{s.t.} & y_i(\mathbf{w}'\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i,\ \xi_i \geq 0 \ \text{ for } \ i = 1, \ldots, n, \end{cases} \tag{1}$$

where $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)'$, and $C > 0$ controls the tradeoff between the margin of the classifier and the total penalty. The Lagrangian dual of problem (1) is

$$\begin{cases} \min_{\boldsymbol{\alpha}} & \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^n \alpha_i \\ \text{s.t.} & \sum_{i=1}^n \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C/n \quad \text{for} \quad i = 1, \ldots, n, \end{cases} \tag{2}$$

where $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)'\phi(\mathbf{x}_j)$ is the kernel function, and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)'$ with α_i being the Lagrangian multiplier for the i-th constraint in Eq. (1). The training examples with nonzero α_i are called support vectors.

The coefficient vector of the classifier in space $\mathcal{H}$ is calculated as

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \phi(\mathbf{x}_i),$$

and the intercept b could be calculated from the support vectors. The obtained classifier is

$$\hat{y} = \mathrm{sign}\left(\sum_{i=1}^n \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right),$$

where $\hat{y}$ is the predicted class label for a new feature vector $\mathbf{x}$.

2.2 Support vector data description

In some practical problems, compared to negative examples, positive examples are relatively easier to obtain and more reliable. As such, instead of fitting a two-class classifier, we can alternatively describe the distribution of positive examples. Toward this end, Tax and Duin (1999, 2004) proposed a support vector data description (SVDD) method, which in the training stage, fits a tight hypersphere in the nonlinear high-dimensional feature space to include most of the training positive examples.

Let the given training dataset be $\{\mathbf{x}_i, i = 1, \ldots, n\}$ with $\mathbf{x}_i \in \mathbb{R}^p$. We assume there is a nonlinear transformation ϕ to transform the feature vector $\mathbf{x}$ to $\phi(\mathbf{x})$, which is in a high-dimensional space $\mathcal{H}$. In this space $\mathcal{H}$, SVDD tries to construct a hypersphere with center $\mathbf{c} \in \mathcal{H}$ and radius $R > 0$ such that the hypersphere contains most of the data and the volume is as small as possible. In other words, the desired hypersphere has the minimum R^2, and at the same time $\|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq R^2$, for $i = 1, \ldots, n$. In addition, as in the SVM formulation, we introduce a set of slack variables $\xi_i \geq 0$, since the training sample might contain outliers. In mathematical form, the problem could be summarized as

$$\min_{R,\mathbf{c},\boldsymbol{\xi}} \quad R^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i, \tag{3}$$

such that

$$\|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq R^2 + \xi_i \ \text{ and } \ \xi_i \geq 0, \quad \text{for} \quad i = 1, \ldots, n, \tag{4}$$

where $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)'$ is the vector of slack variables, and the parameter $C > 0$ controls the tradeoff between the two terms in (3).

The Lagrangian dual of the above optimization problem is

$$\begin{cases} \min_{\boldsymbol{\alpha}} & \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{n}\alpha_i K(\mathbf{x}_i, \mathbf{x}_i) \\ \text{s.t.} & \sum_{i=1}^{n}\alpha_i = 1, \quad 0 \leq \alpha_i \leq \frac{C}{n} \quad \text{for} \quad i = 1, \ldots, n, \end{cases} \tag{5}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)'$ with α_i being the Lagrangian multiplier for the i-th constraint in Eq. (4), and $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function. Once problem (5) is solved, the center of the hypersphere is represented as

$$\mathbf{c} = \sum_{i=1}^{n}\alpha_i\phi(\mathbf{x}_i), \tag{6}$$

and the radius R can be computed from the set of support vectors, i.e., the data points with $\alpha_i \neq 0$.

If the distance from a new example $\mathbf{x}$ to the center $\mathbf{c}$ is less than the radius R, it is classified as a positive example; otherwise, it is classified as a negative example. Thus, the class label for $\mathbf{x}$ is

$$\hat{y} = \text{sign}\left(R^2 - \left\|\phi(\mathbf{x}) - \sum_{i=1}^{n}\alpha_i\phi(\mathbf{x}_i)\right\|^2\right).$$

We note that in support vector clustering (Ben-Hur et al. 2001; Lee and Lee 2005, 2006) algorithms, the same optimization problem as SVDD is first solved and then cluster labels are assigned.

As implemented in popular toolboxes (Chang and Lin 2011; Joachims 1999), the quadratic programming in Eq. (2) and Eq. (5) can be solved by the decomposition methods (Osuna et al. 1997a, b) or sequential minimal optimization (Platt 1998). However, these algorithms are computationally expensive with time complexity about $O(n^3)$ where n is the size of training set. Moreover, the set of support vectors extracted by the quadratic programming is an indicator of the complexity of the obtained model, and is important to estimate the generalization error (Opper and Winther 2000; Vapnik and Chapelle 2000). As such, given the wide applications of SVM/SVDD, other than the expensive quadratic programming, we expect a fast training algorithm which can achieve similar accuracy as the quadratic programming method, and at the same time, it is desired to keep the information of support vectors. The current paper will give an attempt in this direction.

3 The proposed approach

This section first briefly reviews the penalty function method, based on which an approximated formulation to the quadratic programming in Eqs. (2) and (5) is developed, and a generalized Newton algorithm is proposed for SVM and SVDD.

3.1 Penalty function method for optimization problems

In optimization practice, unconstrained problems are often considered to be easier to solve than constrained ones. The idea of penalty function method is to approximate a constrained optimization problem by an unconstrained problem or one with simpler constraints, so that an approximate solution can be obtained in an easier way (Ruszczyński 2006,

chap. 6). For the nonlinear optimization problem

$$\begin{cases} \min_{\mathbf{u}} & f(\mathbf{u}) \\ \text{s.t.} & g_i(\mathbf{u}) \le 0, \quad \text{for} \quad i = 1, \ldots, k; \\ & h_i(\mathbf{u}) = 0, \quad \text{for} \quad i = 1, \ldots, l; \end{cases} \tag{7}$$

the following function is called its quadratic penalty function (Ruszczyński 2006, chap. 6)

$$P_2(\mathbf{u}) = \frac{1}{2}\sum_{i=1}^{k}(g_i(\mathbf{u})_+)^2 + \frac{1}{2}\sum_{i=1}^{l}(h_i(\mathbf{u}))^2 .$$

Clearly, $\mathbf{u}$ satisfies the constraints in problem (7), if and only if $P_2(\mathbf{u}) = 0$.

Consider the unconstrained optimization problem

$$\min_{\mathbf{u}} \ \Phi_\rho(\mathbf{u}) = f(\mathbf{u}) + \rho P_2(\mathbf{u}), \tag{8}$$

where $\rho > 0$. Let the solution to Eq. (8) be $\mathbf{u}_\rho$, then it could be proved that $\mathbf{u}_\rho \to \mathbf{u}^*$ as $\rho \to \infty$, where $\mathbf{u}^*$ is the solution to problem (7). The intuition is that, for ρ sufficiently large, in order to make $\Phi_\rho(\mathbf{u})$ small, $P_2(\mathbf{u})$ should be very close to 0 and at the same time, $f(\mathbf{u})$ should also be as small as possible. In other words, at $\mathbf{u}_\rho$, the constraints are approximately satisfied and the original objective function is small. Thus, we can imagine that as ρ grows, $\mathbf{u}_\rho$ would get closer and closer to $\mathbf{u}^*$. Please see (Ruszczyński 2006,chap. 6) for rigorous proofs to related theoretical results and demonstrative examples.

3.2 Penalty function method for support vector machine

Let $\mathbf{K}$ be the kernel matrix, that is, $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$. Let $\mathbf{H}$ be $n \times n$ matrix with $\mathbf{H}_{ij} = y_i y_j \mathbf{K}_{ij}$. Then, the dual problem for SVM in Eq. (2) could be written compactly in matrix form as

$$\begin{cases} \min_{\boldsymbol{\alpha}} & \frac{1}{2}\boldsymbol{\alpha}'\mathbf{H}\boldsymbol{\alpha} - \mathbf{1}_n'\boldsymbol{\alpha} \\ \text{s.t.} & \mathbf{y}'\boldsymbol{\alpha} = 0 \quad \text{and} \quad \mathbf{0}_n \le \boldsymbol{\alpha} \le \frac{C}{n}\mathbf{1}_n, \end{cases} \tag{9}$$

where $\mathbf{y} = (y_1, y_2, \ldots, y_n)'$. By the penalty function method, problem (9) could be solved approximately via minimizing the following function with respect to $\boldsymbol{\alpha}$,

$$\begin{aligned} F_\rho(\boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{\alpha}'\mathbf{H}\boldsymbol{\alpha} - \mathbf{1}_n'\boldsymbol{\alpha} \\ &\quad + \frac{\rho}{2}\left[(\mathbf{y}'\boldsymbol{\alpha})^2 + \|(-\boldsymbol{\alpha})_+\|^2 + \left\|\left(\boldsymbol{\alpha} - \frac{C}{n}\mathbf{1}_n\right)_+\right\|^2\right] \\ &= \frac{1}{2}\boldsymbol{\alpha}'\mathbf{H}_\rho\boldsymbol{\alpha} - \mathbf{1}_n'\boldsymbol{\alpha} \\ &\quad + \frac{\rho}{2}\left[\|(-\boldsymbol{\alpha})_+\|^2 + \left\|\left(\boldsymbol{\alpha} - \frac{C}{n}\mathbf{1}_n\right)_+\right\|^2\right], \end{aligned} \tag{10}$$

where $\mathbf{H}_\rho = \mathbf{H} + \rho\mathbf{y}\mathbf{y}'$.

For any vector $\mathbf{a} \in \mathbb{R}^n$, there is $\mathbf{a}'\mathbf{y}\mathbf{y}'\mathbf{a} = (\mathbf{a}'\mathbf{y})^2 \ge 0$, that is, the matrix $\mathbf{y}\mathbf{y}'$ is positive semi-definite. Furthermore,

$$\begin{aligned} \mathbf{a}'\mathbf{H}\mathbf{a} &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_i \mathbf{H}_{ij} a_j = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i y_i \mathbf{K}_{ij} y_j a_j \\ &= (\mathbf{a}. * \mathbf{y})'\mathbf{K}(\mathbf{a}. * \mathbf{y}) \ge 0, \end{aligned} \tag{11}$$

where $\mathbf{a}. * \mathbf{y}$ is the vector obtained by component-wise multiplying $\mathbf{a}$ and $\mathbf{y}$, and the inequality in Eq. (11) follows from the positive semi-definiteness of kernel matrix $\mathbf{K}$ (Cortes and Vapnik 1995). Equation (11) shows that $\mathbf{H}$ is a positive semi-definite matrix. Hence, as the sum of two positive semi-definite matrices, $\mathbf{H}_\rho$ is also positive semi-definite.

The gradient vector of $F_\rho(\boldsymbol{\alpha})$ is

$$\nabla F_\rho(\boldsymbol{\alpha}) = \mathbf{H}_\rho\boldsymbol{\alpha} - \mathbf{1}_n - \rho(-\boldsymbol{\alpha})_+ + \rho\left(\boldsymbol{\alpha} - \frac{C}{n}\mathbf{1}_n\right)_+, \tag{12}$$

and the generalized Hessian of $F_\rho(\boldsymbol{\alpha})$ is

$$\partial^2 F_\rho(\boldsymbol{\alpha}) = \mathbf{H}_\rho + \rho\,\mathrm{diag}(-\boldsymbol{\alpha})_* + \rho\,\mathrm{diag}\left(\boldsymbol{\alpha} - \frac{C}{n}\mathbf{1}_n\right)_* . \tag{13}$$

Because $\mathrm{diag}(-\boldsymbol{\alpha})_*$ and $\mathrm{diag}\left(\boldsymbol{\alpha} - \frac{C}{n}\mathbf{1}_n\right)_*$ are diagonal matrices with nonnegative diagonal elements, they are positive semi-definite. Since $\mathbf{H}_\rho$ is positive semi-definite, the generalized Hessian $\partial^2 F_\rho(\boldsymbol{\alpha})$ is a positive semi-definite matrix. This indicates that function $F_\rho(\boldsymbol{\alpha})$ is convex and consequently, it has a minimum point.

To minimize $F_\rho(\boldsymbol{\alpha})$, for its simplicity, we choose to use Newton's method (Boyd and Vandenberghe 2004). In each iteration, Newton algorithm searches for the optimal point in the direction of $-(\nabla^2 F_\rho(\boldsymbol{\alpha}))^{-1}\nabla F_\rho(\boldsymbol{\alpha})$. The regular Hessian of $F_\rho(\boldsymbol{\alpha})$ does not exist because the plus function in $\nabla F_\rho(\boldsymbol{\alpha})$ is not differentiable. Therefore, we use the generalized Hessian of $F_\rho(\boldsymbol{\alpha})$ in the Newton algorithm, since the generalized Hessian has similar properties as the regular Hessian (Hiriart-Urruty et al. 1984). Thus, in the proposed algorithm, we update the solution in the direction of $-(\partial^2 F_\rho(\boldsymbol{\alpha}) + \delta\mathbf{I}_n)^{-1}\nabla F_\rho(\boldsymbol{\alpha})$, and we call the resulting algorithm as generalized Newton algorithm. Here, δ is a small positive number and the term $\delta\mathbf{I}_n$ is added to avoid possible singularity of $\partial^2 F_\rho(\boldsymbol{\alpha})$. The convergence of the generalized Newton algorithm was studied in Mangasarian (2002).

Algorithm 1 summarizes the proposed generalized Newton algorithm for support vector machine (N-SVM).

Algorithm 1: Generalized Newton Algorithm for SVM

0 Initialization: choose a starting point $\boldsymbol{\alpha}^0 \in \mathbb{R}^n$, set the penalty parameter ρ, the perturbation parameter δ, the tolerance level ϵ, the maximum iteration number M. Set the iteration number $t = 0$.
1 Apply Eqs. (12) and (13) to calculate the gradient and the generalized Hessian of F_ρ with current $\boldsymbol{\alpha}^t$.
2 Update $\boldsymbol{\alpha}$ according to $\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t + \gamma_t \mathbf{d}_t$, where $\mathbf{d}_t = -\left(\partial^2 F_\rho(\boldsymbol{\alpha}^t) + \delta \mathbf{I}_n\right)^{-1} \nabla F_\rho(\boldsymbol{\alpha}^t)$ is the search direction, and γ_t is step size.
3 Stop if $\|\boldsymbol{\alpha}^{t+1} - \boldsymbol{\alpha}^t\| < \epsilon$ or $t = M$; otherwise, set $t = t + 1$ and go back to step 1.
4 Return $\boldsymbol{\alpha}^{t+1}$ as the minimum point of F_ρ.

In the second step, the step-size γ_t could be chosen as

$$\gamma_t = \arg\min_{\gamma > 0} \; F_\rho(\boldsymbol{\alpha}^t + \gamma \mathbf{d}_t). \tag{14}$$

The minimization problem in Eq. (14) can be solved by backtracking line search algorithms (Boyd and Vandenberghe 2004, chap. 9). We choose to use Armijo rule (Armijo 1966) for its simplicity, which is given in Algorithm 2 for completeness.

Algorithm 2: Armijo Rule to Determine a Step-size

0 Given the objective function $f(\mathbf{u})$, the current estimation $\mathbf{u}_c$, the current gradient vector $\nabla f(\mathbf{u}_c)$, and the search direction $\mathbf{d}$.
1 Initialize $\gamma = 1$.
2 Calculate $\Delta = f(\mathbf{u}_c + \gamma \mathbf{d}) - f(\mathbf{u}_c)$.
3 If $\Delta \leq \frac{\gamma}{4} \nabla f(\mathbf{u}_c)' \mathbf{d}$, return the current γ as the step size; otherwise, set $\gamma = \gamma/2$, and go back to step 2.

In each iteration of Algorithm 1, we need to calculate a matrix-vector multiplication to determine the search direction in step 2. Since the involved matrix is of size $n \times n$ and the vector is in $\mathbb{R}^n$, the matrix-vector multiplication has time complexity $O(n^2)$. In each iteration of Algorithm 1, we also need to determine the step-size through Algorithm 2, in which we need to evaluate the function value via Eq. (10), and the most expensive calculation is $\boldsymbol{\alpha}' \mathbf{H}_\rho \boldsymbol{\alpha}$, which is again of time complexity $O(n^2)$ since $\mathbf{H}_\rho$ is of size $n \times n$ and the vector $\boldsymbol{\alpha}$ is in $\mathbb{R}^n$. Hence, in each iteration of N-SVM, the time complexity is about $O(n^2)$, and consequently, the total time complexity of N-SVM is about $O(Mn^2)$, where M is the total iteration numbers needed for the algorithm to converge. For large training set, this is significantly more efficient than the quadratic programming-based SVM which has time complexity at the level of $O(n^3)$.

3.3 Penalty function method for support vector data description

Let $\mathbf{k}$ be the $n \times 1$ vector formed by the diagonal elements of the kernel matrix $\mathbf{K}$. Similar to our treatment in Sect. 3.2 for SVM, the dual problem for SVDD in Eq. (5) could be written compactly in the form of matrix as

$$\begin{cases} \min_{\boldsymbol{\alpha}} & \boldsymbol{\alpha}' \mathbf{K} \boldsymbol{\alpha} - \mathbf{k}' \boldsymbol{\alpha} \\ \text{s.t.} & \mathbf{1}_n' \boldsymbol{\alpha} = 1 \quad \text{and} \quad \mathbf{0}_n \leq \boldsymbol{\alpha} \leq \frac{C}{n} \mathbf{1}_n. \end{cases} \tag{15}$$

Same as in Sect. 3.2, applying the penalty function method, we can approximately solve problem (15) by minimizing function

$$\begin{aligned} G_\rho(\boldsymbol{\alpha}) &= \boldsymbol{\alpha}' \mathbf{K} \boldsymbol{\alpha} - \mathbf{k}' \boldsymbol{\alpha} \\ &\quad + \frac{\rho}{2} \left[(\mathbf{1}_n' \boldsymbol{\alpha} - 1)^2 + \|(-\boldsymbol{\alpha})_+\|^2 + \left\| \left(\boldsymbol{\alpha} - \frac{C}{n} \mathbf{1}_n \right)_+ \right\|^2 \right] \\ &= \frac{1}{2} \boldsymbol{\alpha}' \mathbf{K}_\rho \boldsymbol{\alpha} - \mathbf{k}_\rho' \boldsymbol{\alpha} \\ &\quad + \frac{\rho}{2} \left[\|(-\boldsymbol{\alpha})_+\|^2 + \left\| \left(\boldsymbol{\alpha} - \frac{C}{n} \mathbf{1}_n \right)_+ \right\|^2 \right] + \frac{\rho}{2}, \end{aligned} \tag{16}$$

where $\mathbf{k}_\rho = \mathbf{k} + \rho \mathbf{1}_n$ and $\mathbf{K}_\rho = 2\mathbf{K} + \rho \mathbf{1}_n \mathbf{1}_n'$. Similar to Sect. 3.2, we can prove that $\mathbf{K}_\rho$ is a positive semi-definite matrix.

The gradient vector of $G_\rho(\boldsymbol{\alpha})$ is

$$\nabla G_\rho(\boldsymbol{\alpha}) = \mathbf{K}_\rho \boldsymbol{\alpha} - \mathbf{k}_\rho - \rho(-\boldsymbol{\alpha})_+ + \rho \left(\boldsymbol{\alpha} - \frac{C}{n} \mathbf{1}_n \right)_+,$$

and the generalized Hessian of $G_\rho(\boldsymbol{\alpha})$ is

$$\partial^2 G_\rho(\boldsymbol{\alpha}) = \mathbf{K}_\rho + \rho \operatorname{diag}(-\boldsymbol{\alpha})_* + \rho \operatorname{diag} \left(\boldsymbol{\alpha} - \frac{C}{n} \mathbf{1}_n \right)_*.$$

Similar to Sect. 3.2, we can verify that $\partial^2 G_\rho(\boldsymbol{\alpha})$ is a positive semi-definite matrix, which indicates that $G_\rho(\boldsymbol{\alpha})$ is convex and has a minimum point.

Similar to Algorithm 1, we can develop an SVDD algorithm based on generalized Newton's method (N-SVDD), which we choose not to present because it only slightly differs from Algorithm 1. Similar to the analysis presented at the end of Sect. 3.2, we could obtain the computational complexity of N-SVDD as $O(Mn^2)$, where n is the training set size and M is the total number of iterations needed for N-SVDD to converge.

3.4 Discussion

The original formulation of SVM and SVDD leads to a quadratic programming with constraints. By absorbing the constrains, this paper obtains an approximated unconstrained optimization problem, and this will bring some advantages compared to the original quadratic programming. First, the approximated objective function is a quadratic function with positive semi-definite Hessian matrix, which guarantees that there exists a unique minimum point. The solution of the approximated optimization problem converges to the original

quadratic programming solution. Hence, all the information we can get from quadratic programming could also be approximately obtained from the approximated solution. For example, the set of support vectors extracted by both proposed method and quadratic programming are verified very close, see the experimental results in Sect. 4.

Second, the approximated objective function can be minimized by a generalized Newton's method, which is guaranteed to converge quickly (Mangasarian 2002). Theoretical analysis of the quadratic programming-based SVM or SVDD showed that the computational complexity is about $O(n^3)$, which will be verified by our experimental results in Sects. 4.3 to 4.5. However, our theoretical analysis at the end of Sect. 3.2 shows that the computational complexity of N-SVM is about iteration number multiplying $O(n^2)$, which will also be verified by our experimental results. The expensive training cost makes parameter selection for QP-SVM and QP-SVDD not practical, because to select the model parameters, we usually need to use cross validation, which needs to train the model multiple times. On the contrary, due to their efficient training process, it is possible to select parameters for N-SVM and N-SVDD by techniques such as cross validation.

Finally, the Newton's algorithm is easy to implement. The quadratic programming-based SVM and SVDD need to use an external quadratic programming package. For example, in our implementation, we employed a quadratic programming solver developed by about 1000 lines of C++ code. However, from the description of Algorithm 1, we can see that the N-SVM or N-SVDD only needs basic matrix operations, which are built-in functions of almost all modern programming languages. Hence, no specific software package is needed for N-SVM or N-SVDD. For example, in our implementation, N-SVM and N-SVDD each used about 50 lines of MATLAB code. In this sense, we can say that the proposed N-SVM and N-SVDD are much easier to be implemented, compared to the QP counterparts.

In summary, compared to the quadratic programming-based SVM or SVDD, the proposed methods are not only easy to implement, but also more computationally efficient. Moreover, they could extract almost the same set of support vectors as the quadratic programming-based SVM and SVDD. These advantages motivated us to develop this work.

4 Experimental results and analysis

On four pattern classification problems, we compare the performances of the proposed Newton algorithm-based SVM and SVDD (N-SVM and N-SVDD) to those of the ordinary quadratic programming (QP)-based models, i.e., QP-SVM and QP-SVDD.

4.1 Experiment setup and performance measures

All the computer codes were implemented in MATLAB, and the quadratic programming-based models had the QP solver from the C++ version of LIBSVM (Chang and Lin 2011). All the experiments were conducted on a desktop computer with Interl(R) Xeon(R) CPU @ 2.00 GHz and 8 GB memory. During all experiments that incorporated measurement of running time, one core was used solely for the experiments, and the number of other processes running on the system was minimized.

We used the Gaussian kernel

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

with $\sigma = 10$. We set the penalty parameter in SVM and SVDD as $C = 2n$, where n is the training set size. Our purpose is to compare the performances between N-SVM/N-SVDD and QP-SVM/QP-SVDD, and this comparison is fair as long as the parameter settings are the same for the two algorithms because in this case, they solve exactly the same optimization problem with the same parameters. In general, we could select the optimal parameter setting (C, σ) by applying cross validation, generalized approximate cross validation (Wahba et al. 2000), or other criteria mentioned in Chapelle et al. (2002) and Gold and Sollich (2003). Since parameter selection is not the focus of this paper, we choose not to pursue further in this issue. In the generalized Newton algorithm, we set the maximum iteration number to be 1000 and the parameter ρ to be 500, and we found that the result did not differ too much as long as $\rho > 100$. Both the tolerance parameter ϵ and the perturbation parameter δ in Algorithm 1 were set to be 10^{-5}. In all the considered algorithms, $\boldsymbol{\alpha}$ was initialized as $\mathbf{0}_n$.

The receiver operating characteristic (ROC) curve (Fawcett 2006) is employed to illustrate the performance of the classifier. The classifier performs better if the corresponding ROC curve is higher. To numerically compare the ROC curves of different methods, we calculate the area under the curve (AUC) (Fawcett 2006). The bigger the AUC, the better the overall performance of the corresponding classifier.

Similar to the work in Zheng (2019), we use precision and recall rates to compare the support vectors found by the two algorithms. We assume the support vectors extracted by QP-based models as true support vectors, and denote them as the set SV_Q; denote the support vectors from Newton algorithms as SV_N. The precision and recall (Powers 2011) rates are defined as

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

where $|A|$ represents the size of a set A. High precision means that the Newton algorithm finds more correct support vectors than incorrect ones, while high recall means that Newton algorithm extracts most of the correct support vectors.

4.2 Face detection

In the first experiment, we select to use a dataset which consists 5175 face images and 10,000 non-face images. The dataset is available at http://people.missouristate.edu/songfengzheng/FaceData.zip. Each image is of size 16×16, which is normalized so that all pixel values are between 0 and 1. We do not extract any specific features for face detection (e.g., the features used in Viola and Jones 2001), instead, we directly use the pixel values as input to all the considered algorithms. We randomly select 500 face images for training the SVDD models and further randomly select 500 nonface images for training the SVM models, and the remaining images are used for testing purpose.

Fig. 1 gives the ROC curves of the tested algorithms which clearly shows that N-SVM and N-SVDD perform almost the same as their QP counterparts, in the sense that the ROC curves are almost indistinguishable if we plot them in the same figure. Numerically, N-SVM has AUC 0.9907 and QP-SVM has AUC 0.9906; N-SVDD has AUC 0.9241 while QP-SVDD has AUC 0.9353. Thus, both graphical and numerical measures demonstrate that the Newton algorithms and QP based algorithms have almost identical classification performances on the testing set.

We also compare the difference between the final solutions from different algorithms, we have $\|\boldsymbol{\alpha}^{\mathrm{QP}} - \boldsymbol{\alpha}^{\mathrm{N}}\| = 0.0552$ for SVDD models and $\|\boldsymbol{\alpha}^{\mathrm{QP}} - \boldsymbol{\alpha}^{\mathrm{N}}\| = 0.8206$ for SVM, where $\boldsymbol{\alpha}^{\mathrm{QP}}$ is the final solution to Eq. (2) or Eq. (5) and $\boldsymbol{\alpha}^{\mathrm{N}}$ is the final solution of Eq. (10) or Eq. (16), respectively. These differences are reasonably small, considering the size of the problems. This means that the solutions from Newton algorithm and QP based algorithm are quite close.

Training QP-SVM used 106.2333 s, while N-SVM training algorithm converged with only 46 iterations, consuming 1.0108 s, which is about 105 times faster. QP-SVDD used 15.7478 s in training, while N-SVDD training converged in 16 iterations, using 0.2492 s, which is about 63 times faster than the QP counterpart. These numbers indicate that the Newton algorithms are much more time efficient than the QP-based models in training.

QP-SVM found 402 support vectors, while N-SVM extracted 403 support vectors, with the precision rate 99.5% and the recall rate 99.75%. Both QP-SVDD and N-SVDD found the same 26 support vectors, which means that the precision rate and the recall rate are both 100%. The very high precision and recall rates show that Newton algorithms can correctly find almost all the support vectors with very few mistakes, and they also indicate that the Newton algorithms and quadratic programming-based algorithms obtain models with similar complexity.

4.3 Human activity recognition

In the second experiment, we try to recognize human activity (standing, sitting, laying, walking, walking upstairs, walking downstairs) based on inertial sensors for ambient assisted living. The dataset is publicly available from UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones). The training set consists of 4252 data points while the testing set size is 1492, and each data point has 561 features. We normalize the data so that each feature is between 0 and 1.

We use each activity as positive label and use all others as negative label to create six binary classification problems, which are denoted as WK (walking), WU (walking upstairs), WD (walking downstairs), SIT (sitting), STAND (standing), LAY (laying). In training the SVM models, to make the positive and negative examples balance, we further randomly select 500 negative examples for training purpose.

Table 1 compares the performances of QP-SVM and N-SVM. We first notice that the final solutions of QP-SVM ($\boldsymbol{\alpha}^{\mathrm{QP}}$) and those of N-SVM ($\boldsymbol{\alpha}^{\mathrm{N}}$) are close enough, in the sense that the norms of the difference vectors are small, considering the problem size. Table 1 shows that for each problem, the classification performance of QP-SVM and N-SVM are almost identical, in terms of the area under the ROC curve (AUC). We choose not to show the ROC curves because they are almost indistinguishable for each problem. Table 1 also shows that for each problem, N-SVM and QP-SVM found almost the same number of support vectors, obtaining models with similar complexity. However, the training time differs significantly, and N-SVM often converges in 100 iterations.

Table 2 shows the performance comparison between QP-SVDD and N-SVDD. We observe the same pattern as for the SVM classifiers, that is, the final solutions are very similar, the AUC are quite close, with difference below 0.004, both algorithms find almost the same number of support vectors, N-SVDD converges in 20 iterations and is much faster than QP-SVDD in the training phase.

To clearly illustrate the speed advantage of N-SVDD and N-SVM over their QP counterparts, we calculate the ratio between the training times of QP-SVM (QP-SVDD) and N-SVM (N-SVDD) for different problems, and list the results in Table 3. It is clearly seen that, on this dataset, N-SVDD is around 100 times faster than QP-SVDD and N-SVM is 40–130 times faster than QP-SVM.

To study the training time complexity of QP-based models, Table 4 lists $t_{\mathrm{train}}/n^3 \times 10^7$ for QP-SVM and QP-SVDD on the human activity data, where t_{train} is the training time and n is the training set size. We see that the ratio t_{train}/n^3

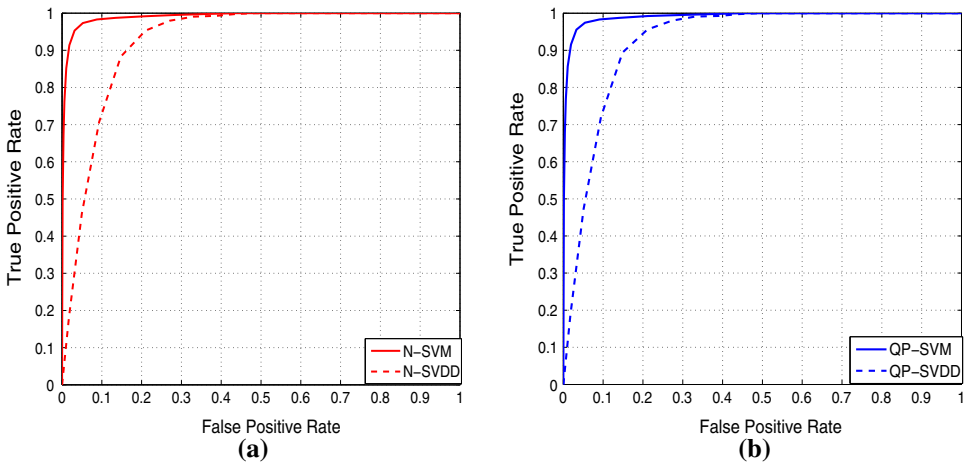

Fig. 1 The ROC curves on the face dataset: **a** Newton algorithm-based classifiers; **b** quadratic programming-based classifiers

Table 1 On the human activity recognition problems, the comparison between QP-SVM and N-SVM

Problem		WK	WU	WD	SIT	STAND	LAY
Training set size		1269	1129	1191	1334	1275	1054
Iter # N-SVM		65	49	48	155	105	68
$\|\boldsymbol{\alpha}^{\mathrm{QP}} - \boldsymbol{\alpha}^{\mathrm{N}}\|$		0.3906	0.5279	0.6231	0.3238	0.3229	0.2484
AUC	QP-SVM	0.8747	0.7481	0.8650	0.8375	0.8719	0.9765
	N-SVM	0.8749	0.7482	0.8649	0.8375	0.8720	0.9765
# SV	QP-SVM	694	806	723	714	787	691
	N-SVM	694	806	722	710	784	688
Time	QP-SVM	209.7850	145.7431	173.8629	253.3498	216.3342	122.8166
	N-SVM	3.2364	1.2047	1.3188	6.2629	3.7418	1.5510

The listed are the closeness of the solutions evaluated by $\|\boldsymbol{\alpha}^{\mathrm{QP}} - \boldsymbol{\alpha}^{\mathrm{N}}\|$, the area under ROC curve (AUC), the number of support vectors (# SV), and the training times (in seconds). The training set sizes and the iteration numbers for N-SVM to converge are also given

Table 2 On the human activity recognition problems, the comparison between QP-SVDD and N-SVDD

Problem		WK	WU	WD	SIT	STAND	LAY
Training set size		769	629	691	834	775	554
Iter # N-SVDD		19	19	18	17	18	15
$\|\boldsymbol{\alpha}^{\mathrm{QP}} - \boldsymbol{\alpha}^{\mathrm{N}}\|$		0.0031	0.0042	0.0058	0.0017	0.0012	0.0011
AUC	QP-SVDD	0.5533	0.7004	0.6132	0.7003	0.8411	0.7798
	N-SVDD	0.5516	0.6971	0.6114	0.7011	0.8407	0.7794
# SV	QP-SVDD	4	9	5	9	11	11
	N-SVDD	4	8	5	9	11	11
Time	QP-SVDD	61.6968	33.4195	45.2624	80.2963	66.2835	23.2798
	N-SVDD	0.5560	0.3554	0.3777	0.5986	0.5885	0.2412

See the caption of Table 1 for more information

Table 3 The ratio of training times of QP-SVDD vs. N-SVDD and QP-SVM vs. N-SVM on different problems

Problem	WK	WU	WD	SIT	STAND	LAY
SVDD	110.9691	94.0463	119.8468	134.1491	112.6274	96.5242
SVM	64.8210	120.9797	131.8382	40.4526	57.8160	79.1841

Table 4 The time complexity of QP-SVM and QP-SVDD on the human activity recognition problems

Problem	WK	WU	WD	SIT	STAND	LAY
QP-SVM	1.027	1.013	1.029	1.067	1.044	1.049
QP-SVDD	1.357	1.343	1.372	1.384	1.424	1.369

The listed are the values of $t_{\text{train}}/n^3 \times 10^7$, where n is the training set size

Table 5 The time complexity of N-SVM and N-SVDD on the human activity recognition problems

Problem	WK	WU	WD	SIT	STAND	LAY
N-SVM	3.092	1.929	1.937	2.271	2.192	2.053
N-SVDD	4.948	4.728	4.395	5.062	5.443	5.239

The listed are the values of $t_{\text{train}}/(Mn^2) \times 10^8$, where n is the training set size and M is the iteration numbers needed for N-SVM and N-SVDD to converge

for QP-SVM is roughly 1.0×10^{-7}, and for QP-SVDD, it is close to 1.4×10^{-7}. These results indicate that the time complexity of QP-SVM and QP-SVDD is about $O(n^3)$.

To verify our theoretical analysis of the time complexity of Newton algorithm-based SVM and SVDD presented at the end of Sect. 3.2, Table 5 presents $t_{\text{train}}/(Mn^2) \times 10^8$ for N-SVM and N-SVDD on the human activity recognition problems, where M is the iteration numbers needed. We see that the ratio $t_{\text{train}}/(Mn^2)$ for N-SVM is roughly 2×10^{-8}, and for N-SVDD, it is close to 5×10^{-8}. These results support our analysis that the time complexity of N-SVM and N-SVDD is about $O(Mn^2)$.

To investigate the overlap of the sets of support vectors extracted by different methods, we calculate the precision and recall rates for each problem, given in Table 6. The results show that all the precision rates are 100%, which means that N-SVM and N-SVDD do not miss any single support vector which was found by their QP counterparts, and all but one

Table 6 On the human activity recognition problems, the precision (P) and recall (R) rates for the support vectors extracted by N-SVM and N-SVDD

Problem		WK (%)	WU (%)	WD (%)	SIT (%)	STAND (%)	LAY (%)
N-SVM	P	100	100	100	100	100	100
	R	100	100	99.86	99.44	99.62	99.57
N-SVDD	P	100	100	100	100	100	100
	R	100	88.89	100	100	100	100

Table 7 On the handwritten digit recognition problems, the comparison between QP-SVM and N-SVM

Problem		"2"	"3"	"4"	"5"	"6"
Training set size		1731	1658	1652	1556	1664
Iter # N-SVM		30	30	35	26	31
$\|\boldsymbol{\alpha}^{\text{QP}} - \boldsymbol{\alpha}^{\text{N}}\|$		0.7199	0.6087	1.0417	0.4089	0.7667
AUC	QP-SVM	0.9798	0.9850	0.9863	0.9881	0.9967
	N-SVM	0.9801	0.9851	0.9859	0.9881	0.9966
# SV	QP-SVM	355	330	338	400	255
	N-SVM	358	330	341	400	257
Time	QP-SVM	578.433	538.172	508.088	429.723	529.014
	N-SVM	1.8557	1.7576	1.8653	1.2176	1.7100

See the caption of Table 1 for more information

Table 8 On the handwritten digit recognition problems, the comparison between QP-SVDD and N-SVDD

Problem		"2"	"3"	"4"	"5"	"6"
Training set size		731	658	652	556	664
Iter # N-SVDD		12	12	13	12	13
$\|\boldsymbol{\alpha}^{\text{QP}} - \boldsymbol{\alpha}^{\text{N}}\|$		0.0130	0.0137	0.0127	0.0135	0.0099
AUC	QP-SVDD	0.8244	0.9358	0.9409	0.7769	0.9674
	N-SVDD	0.8254	0.9371	0.9427	0.7772	0.9695
# SV	QP-SVDD	35	26	26	30	22
	N-SVDD	38	29	29	31	23
Time	QP-SVDD	57.4966	38.1632	37.7701	22.4223	40.5231
	N-SVDD	0.3252	0.2248	0.2490	0.1506	0.3042

See the caption of Table 1 for more information

Table 9 The ratio of training times of QP-SVDD vs. N-SVDD and QP-SVM vs. N-SVM on the handwritten digit recognition problems

Problem	"2"	"3"	"4"	"5"	"6"
SVDD	188.6468	151.3330	135.7874	127.5034	150.4354
SVM	311.7111	306.2012	272.3854	352.9354	309.3564

recall rates are above 99.5%, which indicates that N-SVM and N-SVDD extract almost all the support vectors found by the QP-based algorithms.

4.4 Handwritten digit recognition

In this experiment, we use a dataset which consists of normalized handwritten digits ("0" to "9") of size 16×16. The dataset has 7291 training examples and 2007 testing examples, and is available at https://web.stanford.edu/~hastie/ElemStatLearn/. Same as the experiment in Sect. 4.2, we simply use these 256 pixel values as inputs to the algorithms. We create five binary classification problems, with digits "2", "3", "4", "5", and "6" as positive label, respectively; in each problem, we use all other nine digits as negative label. In training the SVM models, to make the positive and negative examples balance, we further randomly select 1000 negative examples.

Similar to the experiments presented in Sect. 4.3, Tables 7 and 8 compare the performances of QP-SVM vs. N-SVM and QP-SVDD vs. N-SVDD on handwritten digit recognition problems, respectively. We could draw the same conclusion as in Sect. 4.3 that Newton algorithm-based SVM/SVDD found almost the same number of support vectors as their QP-based counterparts, and N-SVM/N-SVDD have almost identical classification performances as QP-SVM/QP-SVDD in terms of AUC. However, the Newton algorithm-based models are much faster in the training phase. Also, we see that N-SVM and N-SVDD converge in 40 iterations. Table 9 presents the ratio between the training times of QP-SVM (QP-SVDD) and N-SVM (N-SVDD) for different problems. It is clearly seen that, on this dataset, N-SVDD is more than 120 times faster than QP-SVDD and N-SVM is 200–300 times faster than QP-SVM.

Table 10 lists $t_{\text{train}}/n^3 \times 10^7$ for QP-SVM and QP-SVDD on the handwritten digit recognition problems. Same as in Sect. 4.3, these results indicate that the time complexity of QP-SVM and QP-SVDD is about $O(n^3)$. Table 11 presents $t_{\text{train}}/(Mn^2) \times 10^8$ for N-SVM and N-SVDD on the tested tasks. We see that the ratio $t_{\text{train}}/(Mn^2)$ for N-SVM is roughly 2×10^{-8}, and of N-SVDD, it is close to 4.5×10^{-8}. These results verify that the time complexity of N-SVM and N-SVDD is about $O(Mn^2)$, consistent with the theoretical analysis at the end of Sect. 3.2.

Table 12 shows the precision and recall rates for the support vectors extracted by N-SVM and N-SVDD, comparing to their QP counterparts. We could get the same conclusion as in the experiment in Sect. 4.3, that is, N-SVM/N-SVDD extract almost the same set of support vectors as QP-SVM/QP-SVDD.

Table 10 The time complexity of QP-SVM and QP-SVDD on the handwritten digit recognition problems

Problem	"2"	"3"	"4"	"5"	"6"
QP-SVM	1.115	1.181	1.127	1.141	1.148
QP-SVDD	1.472	1.340	1.363	1.305	1.384

Please see the caption of Table 4 for more information

Table 11 The time complexity of N-SVM and N-SVDD on the handwritten digit recognition problems

Problem	"2"	"3"	"4"	"5"	"6"
N-SVM	2.064	2.131	1.953	1.934	1.992
N-SVDD	5.071	4.327	4.506	4.060	5.307

Please see the caption of Table 5 for more information

Table 12 On the handwritten digit recognition problems, the precision (P) and recall (R) rates for the support vectors extracted by N-SVM and N-SVDD

Problem		"2" (%)	"3" (%)	"4" (%)	"5" (%)	"6" (%)
N-SVM	P	99.16	100	99.12	100	99.22
	R	100	100	100	100	100
N-SVDD	P	92.11	89.66	89.66	96.77	95.65
	R	100	100	100	100	100

4.5 Music genre classification

To test the algorithms on different modalities of data, we ran the program on a music dataset (Defferrard et al. 2017) to recognize different genres of the music. The dataset was downloaded from https://github.com/mdeff/fma, which includes 106,574 tracks of 30 s mp3 files, with 161 unbalanced genres. Each data point has 518 features which are extracted using Python library *librosa*. We select "Rock", "Experimental", and "Electronic" as genres to be classified, resulting in 33665 data points, with 14,174 for "Rock", 10,119 for "Experimental", and 9372 for "Electronic".

We create three classification problems, each having one genre as positive label and the other two genres as negative label. For each problem, we randomly select 2500 positive

Table 13 On the music genre classification problems, the comparison between QP-SVM and N-SVM

Problem		"Rock"	"Experimental"	"Electronic"
Training set size		5000	5000	5000
Iter # N-SVM		647	753	333
$\|\boldsymbol{\alpha}^{\mathrm{QP}} - \boldsymbol{\alpha}^{\mathrm{N}}\|$		0.9942	0.9899	1.0417
AUC	QP-SVM	0.6329	0.6270	0.6632
	N-SVM	0.6680	0.6270	0.6633
# SV	QP-SVM	4998	4997	4904
	N-SVM	4783	4997	4901
Time	QP-SVM	11189.2055	12997.3853	11236.1094
	N-SVM	396.7772	461.3587	204.0731

See the caption of Table 1 for more information

Table 14 On the music genre classification problems, the comparison between QP-SVDD and N-SVDD

Problem		"Rock"	"Experimental"	"Electronic"
Training set size		2500	2500	2500
Iter # N-SVDD		30	31	34
$\|\boldsymbol{\alpha}^{\mathrm{QP}} - \boldsymbol{\alpha}^{\mathrm{N}}\|$		0.0068	0.0139	0.0076
AUC	QP-SVDD	0.6424	0.3182	0.5226
	N-SVDD	0.6421	0.3182	0.5226
# SV	QP-SVDD	37	44	36
	N-SVDD	37	44	36
Time	QP-SVDD	1900.0605	1920.0495	1906.7367
	N-SVDD	7.3111	7.6771	8.3348

See the caption of Table 1 for more information

examples to train the QP-SVDD and N-SVDD models; to train QP-SVM and N-SVM, we randomly add 2500 negative examples to the training set, to balance the positive and negative examples. Due to the limitation of available computing resource, we did not test problems with size beyond 5000.

Similar to the experiments presented in Sects. 4.3 and 4.4, Tables 13 and 14 compare the performances of QP-SVM vs. N-SVM and QP-SVDD vs. N-SVDD on the music genre classification problems, respectively. We could observe that, compared to their QP counterparts, N-SVM and N-SVDD have almost identical classification performances in terms of AUC, and they could find almost the same number of support vectors as QP-SVM/QP-SVDD. However, the Newton algorithm-based models use significantly less time in the training phase. Table 15 presents the ratio between the training times of QP-SVM (QP-SVDD) and N-SVM (N-SVDD) for different problems, which clearly demonstrates that, on this dataset, N-SVDD is more than 200 times faster than QP-SVDD and N-SVM is 20–60 times faster than QP-SVM.

Table 16 lists $t_{\mathrm{train}}/n^3 \times 10^7$ for QP-SVM and QP-SVDD on the music genre classification problem, and Table 17 presents $t_{\mathrm{train}}/(Mn^2) \times 10^8$ for N-SVM and N-SVDD on the same problem. Same as in Sects. 4.3 and 4.4, these results indicate that the time complexity of QP-SVM and QP-SVDD is about $O(n^3)$ and the time complexity of N-

Table 15 The ratio of training times of QP-SVDD vs. N-SVDD and QP-SVM vs. N-SVM on the music genre classification problems

Problem	"Rock"	"Experimental"	"Electronic"
SVDD	259.8861	250.1000	228.7684
SVM	28.2002	28.1720	55.0592

Table 16 The time complexity of QP-SVM and QP-SVDD on the music genre classification problems

Problem	"Rock"	"Experimental"	"Electronic"
QP-SVM	0.8951	1.0398	0.8989
QP-SVDD	1.2160	1.2288	1.2203

Please see the caption of Table 4 for more information

Table 17 The time complexity of N-SVM and N-SVDD on the music genre classification problems

Problem	"Rock"	"Experimental"	"Electronic"
N-SVM	2.4530	2.4508	2.4513
N-SVDD	3.8993	3.9624	3.9223

Please see the caption of Table 5 for more information

Table 18 On the music genre classification problems, the precision (P) and recall (R) rates for the support vectors extracted by N-SVM and N-SVDD

Problem		"Rock" (%)	"Experimental" (%)	"Electronic" (%)
N-SVM	P	99.98	100.00	100.00
	R	95.68	100.00	99.94
N-SVDD	P	100.00	100.00	100.00
	R	100.00	100.00	100.00

SVM and N-SVDD is about $O(Mn^2)$, consistent with the theoretical analysis in Sect. 3.2. Table 18 shows the precision and recall rates for the support vectors extracted by N-SVM and N-SVDD, comparing to their QP counterparts. As in Sects. 4.3 and 4.4, we could conclude that N-SVM/N-SVDD extract almost the same set of support vectors as QP-SVM/QP-SVDD.

In summary, all our experiments show that, compared to QP-SVM and QP-SVDD, the proposed N-SVM and N-SVDD have almost identical classification performances, in terms of ROC analysis. Furthermore, N-SVM and N-SVDD could get almost the same set of support vectors as their QP counterparts, indicating that the models obtained by both methods have similar complexity. However, our experimental results show that N-SVM and N-SVDD are much more time-efficient in training. More importantly, we should mention that in our implementation, the core quadratic programming code for QP-SVM and QP-SVDD was developed in C++ which is much more computationally efficient than MATLAB, in which N-SVM and N-SVDD were implemented. Taking this factor into account, the proposed N-SVM and N-SVDD would be much more time-efficient than QP-SVM and QP-SVDD, if they were implemented in the same programming language and ran on the same platform.

5 Conclusion and future works

The formulations of SVM and SVDD lead to a quadratic programming which is computationally expensive to solve. This paper proposes an alternative approach to solve the dual optimization problem. We first apply the idea of quadratic penalty function method to incorporate the constraints to the objective function, resulting in an unconstrained minimization problem. Then, a generalized Newton algorithm is employed with Armijo search for step length. The resulting algorithms are referred to as Newton SVM (N-SVM) and Newton SVDD (N-SVDD), which are easy to implement, without requiring any additional sophisticated toolbox other than standard matrix operations.

Extensive experiments were conducted on various pattern classification problems, and we compared the performance of the proposed N-SVM/N-SVDD to that of QP-SVM/QP-SVDD, in terms of ROC curve analysis, support vectors extracted by the two methods, and the training time. All our results show that the developed N-SVM and N-SVDD have similar classification performance to their QP counterparts, but much more efficient in training. On the tested problems, N-SVM is tens to hundreds of times faster than QP-SVM, and N-SVDD is hundreds of times faster than QP-SVDD. Furthermore, the two methods extract almost identical set of support vectors, indicating that the obtained models have similar complexity.

We notice that the idea presented in this paper can be applied to other variants of support vector models, for example, the asymmetric SVM model (Bach et al. 2006), SVM with different loss functions (Huang et al. 2014), SVDD with both positive and negative examples (Tax and Duin 2004), support vector clustering (Ben-Hur et al. 2001; Lee and Lee 2005, 2006), and support vector regression (Smola and Schölkopf 2002), because all of these models are formulated as solving a quadratic programming.

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

Funding This work was supported by a Summer Faculty Fellowship from Missouri State University.

Declarations

Conflict of interest The author certifies that there is no conflicts of interest or competing interest.

References

Armijo L (1966) Minimization of functions having Lipschitz-continuous first partial derivatives. Pac J Math 16:1–3

Bach FR, Heckerman D, Horvitz E (2006) Considering cost asymmetry in learning classifiers. J Mach Learn Res 7:1713–1741

Ben-Hur A, Horn D, Siegelmann HT, Vapnik V (2001) Support vector clustering. J Mach Learn Res 2:125–137

Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press, Cambridge

Chang CC, Lin CJ (2011) LIBSVM: a library for support vector machines. ACM Trans Intell Syst Technol 2(3):1–27

Chapelle O (2007) Training a support vector machine in the primal. Neural Comput 19:1155–1178

Chapelle O, Vapnik V, Bousquet O, Mukherjee S (2002) Choosing multiple parameters for support vector machines. Mach Learn 46(1–3):131–159

Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297

Defferrard M, Benzi K, Vandergheynst P, Bresson X (2017) FMA: a dataset for music analysis. In: Proceedigns of 18th International Society for Music Information Retrieval Conference (ISMIR)

Gold C, Sollich P (2003) Model selection for support vector machine classification. Neurocomputing 55(1–2):221–249

Fawcett T (2006) An introduction to ROC analysis. Pattern Recognit Lett 27(8):861–874

Hiriart-Urruty J-B, Strodiot JJ, Nguyen VH (1984) Generalized Hessian matrix and second-order optimality conditions for problems with $C^{1,1}$ data. Appl Math Optim 11(1):43–56

Huang X, Shi L, Suykens JAK (2014) Support vector machine classifier with pinball loss. IEEE Trans PAMI 36(5):984–997

Lee J, Lee D (2005) An improved cluster labeling method for support vector clustering. IEEE Trans PAMI 27(3):461–464

Lee J, Lee D (2006) Dynamic characterization of cluster structures for robust and inductive support vector clustering. IEEE Trans PAMI 28(11):1869–1874

Lee YJ, Mangasarian OL (2001) SSVM: a smooth support vector machine. Comput Optim Appl 20:5–22

Joachims J (1999) Making large-scale SVM learning practical. In: Advances in kernel methods—support vector learning. MIT-Press, Cambridge

Mangasarian OL (2002) A finite newton method for classification. Optim Methods Softw 17:913–929

Opper M, Winther O (2000) Gaussian process and SVM: mean field and leave-one-out. In: Advances in large margin classifiers. MIT Press, Cambridge, pp 261– 280

Osuna E, Freund R, Girosi F, (1997a) An improved training algorithm for support vector machines. In: Proc. of IEEE workshop neural networks for signal processing, pp 276 – 285

Osuna E, Freund R, Girosi F (1997b) Training support vector machines: an application to face detection. In: Proc, IEEE CVPR

Platt J (1998) Fast training of support vector machines using sequential minimal optimization. In: Advances in kernel methods—support vector learning. MIT-Press, Cambridge

Powers DMW (2011) Evaluation: from precision, recall and F-measure to ROC, informedness, markedness & correlation. J Mach Learn Technol 2(1):37–63

Ruszczyński A (2006) Nonlinear optimization. Princeton University Press, Princeton

Shalev-Schwartz S, Singer Y, Srebro N, Cotter A (2011) Pegasos: Primal estimated sub-gradient solver for SVM. Math Program 127(1):3–30

Smola AJ, Schölkopf B (2002) A tutorial on support vector regression. Stat Comput 14(3):199–222

Tax DMJ, Duin RPW (1999) Support vector domain description. Pattern Recognit Lett 20(11–13):1191–1199

Tax DMJ, Duin RPW (2004) Support vector data description. Mach Learn 54(1):45–66

Vapnik V,Chapelle O (2000) Bounds on error expectation for SVM. In: Advances in large margin classifiers. MIT Press, Cambridge, pp 311–326

Viola P, Jones M (2001) Rapid Object detection using a boosted cascade of simple features. In: Proc. of IEEE CVPR

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: Advances in large margin classifiers. MIT Press, Cambridge

Wang Z, Crammer K, Vucetic S (2012) Breaking the curse of kernelization: budgeted stochastic gradient descent for large-scale SVM training. J Mach Learn Res 13(1):3103–3131

Zheng S (2016) Smoothly approximated support vector domain description. Pattern Recogn 49(1):55–64

Zheng S (2019) A fast iterative Algorithm for support vector data description. Int J Mach Learn Cybern 10(5):1173–1187