Neural Computing and Applications (2023) 35:6095–6113
https://doi.org/10.1007/s00521-022-07875-1

ORIGINAL ARTICLE

A fast conjugate functional gain sequential minimal optimization training algorithm for LS-SVM model

Lang Yu[1] · Xin Ma[2] · Shengjie Li[1]

Received: 12 March 2022 / Accepted: 21 September 2022 / Published online: 15 November 2022

Abstract
The least squares support vector machine (LS-SVM) is an effective method to deal with classification and regression problems and has been widely studied and applied in the fields of machine learning and pattern recognition. The learning algorithms of the LS-SVM are usually conjugate gradient (CG) and sequential minimal optimization (SMO) algorithms. Based on this, we propose a conjugate functional gain SMO algorithm and theoretically prove its asymptotic convergence. This algorithm combines the conjugate direction method and the functional gain SMO algorithm with second-order information, which increases the functional gain of the plain SMO algorithm. In addition, we also provide a generalized SMO-type algorithm framework with a simple iterative format and easy implementation for other LS-SVM training algorithms. The numerical results show that the execution time of this algorithm is significantly shorter than that of the other plain SMO-type algorithms and CG-type algorithms.

Keywords LS-SVM model · Sequential minimal optimization · Conjugate direction · Functional gain

1 Introduction

Support vector machines (abbreviated as SVMs) are an important class of methods to solve classification [1] and regression problems [2]. When it solves linear inseparable and nonlinear regression problems, it usually uses kernel tricks to map data from a low-dimensional input space to a high-dimensional space. SVMs have always had a place in the field of pattern recognition with their elegant mathematical optimization theory and excellent data prediction ability. After continuous research, the theory and application of SVMs have been well developed [3]. One of the most successful variants is the least squares support vector machine (abbreviated as LS-SVM). This model is also a powerful tool for handling classification and regression problems in machine learning.

The LS-SVM was first proposed by Suykens et al. [4]. Its primal problem is a convex optimization problem with a linear equality constraint. To make the training of the LS-SVM more efficient, researchers have proposed fast training algorithms. Jiao et al. [5] used an approximate algorithm to quickly train the LS-SVM and improve its sparsity. Yang et al. [6] proposed to use a pruning algorithm to effectively solve the optimization problem of the LS-SVM. Li et al. [7] proposed a fast iterative single-data method for training unconstrained LS-SVM. Xia [8] used QR factorization to train sparse LS-SVM. Chua [9] proposed a computationally efficient method for solving large-scale LS-SVM classifiers based on the Sherman–Morrison–Woodbury (abbreviated as SMW) matrix. However, the algorithm has high requirements on the kernel mapping, and the concrete form of kernel mapping must be given. Suykens et al. [10] used the conjugate gradient (abbreviated as CG) method to solve the Karush–Kuhn–Tucker (abbreviated as KKT) equations of the LS-SVM. However, in their method, the CG algorithm needs to be used twice. Hence, Chu et al. [11] proposed an improved CG

✉ Shengjie Li
lisj@cqu.edu.cn

Lang Yu
ylang@cqu.edu.cn

Xin Ma
cauchy7203@gmail.com

[1] College of Mathematics and Statistics, Chongqing University, Chongqing 401331, China

[2] School of Science, Southwest University of Science and Technology, Mianyang 621010, China

(abbreviated as ICG) method to solve the KKT equations. This method only needs to use the CG method once and reduces the order of the KKT equations. Li et al. [12] transformed the equality constraint problem in the LS-SVM into an unconstrained optimization problem and proposed to use recursive formula and CG method to quickly train the LS-SVM. In general, for large-scale datasets, it is more efficient to use the sequential minimal optimization (abbreviated as SMO) algorithm to solve the LS-SVM [12].

The SMO algorithm was proposed by Platt [13] and used to train the standard SVMs. This method takes the decomposition algorithm to the extreme, and only two variables are selected for iteration at a time. For each subproblem, the SMO algorithm can obtain the analytical solution, avoiding the complex matrix computations of quadratic programming numerical algorithm. The SMO algorithm is not only effective for classification problems but also efficient for regression problems [2]. Keerthi et al. [14] first proposed a SMO algorithm for solving the LS-SVM. This algorithm also uses maximal violating pair (abbreviated as MVP) to select the working set and is also called the first-order SMO. In SVMs, the second-order SMO is more efficient than the first-order SMO [15]. Hence, Lopez et al. [16] introduced the second-order SMO into the LS-SVM to improve the training speed. Shao et al. [17] ignored the bias term of the LS-SVM and proposed a single-direction SMO algorithm (abbreviated as SD-SMO). Without considering the bias term, the efficiency of the SD-SMO is higher than that of the first-order SMO. Bo et al. [18] proposed the functional gain SMO algorithm (abbreviated as FGSMO) and verified its efficiency with numerical experiments. And FGSMO can degenerate into the second-order SMO. In general, parallel computing [19] and distributed algorithms can also improve the learning efficiency of the model, for instance, parallel SMO [20–25] and ADMM algorithms [26, 27], etc. However, they mostly divide the dataset and assign them to different processors.

Alternatively, the SMO algorithm can also be regarded as a special kind of projected gradient [28]. Naturally, some gradient acceleration tricks can also be applied to SMO-type algorithms. For instance, Lopez et al. [29] combined momentum and SMO algorithm to propose a momentum-accelerated SMO algorithm (abbreviated as MLS-SMO). Torres-Barrán et al. [30] used the Nestrove acceleration strategy on the SMO algorithm, reducing the number of iterations of the SMO algorithm. However, among the optimization algorithms, the conjugate direction method is also an important class of unconstrained optimization algorithms. It not only speeds up the convergence speed of the gradient descent algorithm, but also avoids a large number of numerical computation of Newton's method. It is a relatively practical and effective optimization algorithm between gradient descent and Newton's method. Hence, we will develop a new conjugate variant SMO (abbreviated as CSMO) algorithm for LS-SVM. Although the CSMO was originally used to train SVMs [28], from the mathematical model of LS-SVM and SVMs, LS-SVM is more suitable. Because the new CSMO algorithm for LS-SVM will not be affected by the box constraints, and there is no need to set a restart step, the iterative format will be simpler. In addition, the algorithm usually has a larger functional gain than the plain SMO-type algorithms and can converge to the optimal solution of the specified accuracy at a faster speed. Especially when the penalty parameter is large, the efficiency of the new CSMO of LS-SVM will be significantly higher than other plain SMO-type algorithms. Hence, we call it the conjugate functional gain SMO algorithm (abbreviated as CFGSMO)

The rest of this work is organized as follows. In Sect. 2, we will briefly introduce the LS-SVM. The solution algorithms for LS-SVM are summarized in Sect. 3. The CFGSMO algorithm is proposed in Sect. 4. The convergence analysis of the CFGSMO is discussed in Sect. 5. The numerical experiments are shown in Sect. 6. The last section is the summary of the work. Table 1 is the notation table.

2 Preliminaries of the LS-SVM model

The mathematical theory of LS-SVM and its connection with the system of linear equations will be briefly introduced in this section.

Consider a given sample training set $\boldsymbol{T} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathbb{R}^p$ is the i-th pattern input vector, and n is the number of samples. In the binary classification problem, category label $y_i \in \{-1, 1\}$. At this time, the LS-SVM needs to solve a quadratic programming problem with linear equality constraints, that is

$$\begin{aligned} &\min_{\boldsymbol{\omega},b,\boldsymbol{\xi}} \boldsymbol{P}(\boldsymbol{\omega}, b, \boldsymbol{\xi}) = \frac{1}{2}\boldsymbol{\omega}^T\boldsymbol{\omega} + \frac{\gamma}{2}\sum_{i=1}^{n}\xi_i^2 \\ &s.t. \qquad \boldsymbol{\omega}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b = y_i - \xi_i, \forall i, \end{aligned} \tag{1}$$

where the $\boldsymbol{\phi} : \Omega \rightarrow \boldsymbol{H}$ is a kernel mapping from the input space $\Omega(\Omega \subset \mathbb{R}^p)$ to the high-dimensional feature space $\boldsymbol{H}$. The purpose of introducing the kernel mapping $\boldsymbol{\phi}$ is to transform the nonlinear problem in the input space into the linear problem in the high-dimensional space. $\boldsymbol{\omega}$ is the weight parameter, b is the bias term, γ is the penalty parameter, and ξ_i is the error variable. When $y_i \in \mathbb{R}$, LS-

Table 1 Notation table

Notation	Instruction	Notation	Instruction
$\boldsymbol{\omega}$	Weight vector	$\boldsymbol{\xi}$	Error variable
γ	Penalty parameter	$\boldsymbol{\phi}$	Kernel mapping
b	Bias	$\boldsymbol{\alpha}, v$	Lagrangian multiplier
$\boldsymbol{G}(\boldsymbol{\alpha})$	The gradient of $\mathcal{D}(\boldsymbol{\alpha})$	$\boldsymbol{K}, \widetilde{\boldsymbol{K}}$	Kernel matrix
$\boldsymbol{I}$	Identity matrix	n	Sample size
$\Omega, \boldsymbol{H}$	Space	ρ	Step length
ϵ	Tolerance parameter	$\boldsymbol{s}$	Conjugate direction
$\boldsymbol{h}$	Search direction of the SMO	$\boldsymbol{T}$	Dataset
$\boldsymbol{x}_i$	Input pattern vector	y_i	Category label
(i,j)	Working set	σ	Kernel width

SVM is used to deal with the regression problems. The Lagrangian function of problem (1) is

$$\begin{aligned}\mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) =& \frac{1}{2}\boldsymbol{\omega}^T\boldsymbol{\omega} + \frac{\gamma}{2}\sum_{i=1}^{n}\xi_i^2 \\ &- \sum_{i=1}^{n}\alpha_i(\boldsymbol{\omega}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b + \xi_i - y_i),\end{aligned} \tag{2}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^n$ is the Lagrangian multiplier. According to the KKT condition of problem (1), we can get

$$\begin{aligned}&\nabla_{\omega}\mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = 0 \Longrightarrow \boldsymbol{\omega} = \sum_{i=1}^{n}\alpha_i\boldsymbol{\phi}(\boldsymbol{x}_i) \\ &\nabla_{b}\mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = 0 \Longrightarrow \sum_{i=1}^{n}\alpha_i = 0 \\ &\nabla_{\xi_i}\mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = 0 \Longrightarrow \alpha_i = \gamma\xi_i, i = 1, 2, \ldots, n \\ &\nabla_{\alpha_i}\mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = 0 \Longrightarrow \boldsymbol{\omega}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b + \xi_i - y_i = 0, i = 1, 2, \ldots, n.\end{aligned} \tag{3}$$

After eliminating the weight variable $\boldsymbol{\omega}$ and the error variable ξ_i, (3) can be further expressed as the linear equation system

$$\begin{pmatrix} 0 & \boldsymbol{y}^T \\ \boldsymbol{y} & \boldsymbol{K} + \gamma^{-1}\boldsymbol{I} \end{pmatrix}\begin{pmatrix} b \\ \boldsymbol{\alpha} \end{pmatrix} = \begin{pmatrix} 0 \\ \boldsymbol{1} \end{pmatrix}, \tag{4}$$

where $[\boldsymbol{K}]_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle\boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j)\rangle$ is the Gram matrix, $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the kernel function that satisfies the Mercer condition, $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^T$, and $\boldsymbol{1}_{n\times 1} = (1, 1, \ldots, 1)^T$. The matrix $\boldsymbol{K} + \gamma^{-1}\boldsymbol{I}$ is a symmetric positive definite matrix, since matrix $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ is a positive semi-definite matrix and γ is always positive, the diagonal of $\gamma^{-1}\boldsymbol{I}$ is positive.

If $\boldsymbol{A}$ is the coefficient matrix of (4), then the matrix $\boldsymbol{A} \in \mathbb{R}^{(n+1)\times(n+1)}$. When the sample $\boldsymbol{T} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ is large, the dimension of the matrix $\boldsymbol{A}$ is large and storage is difficult. At this time, it is not particularly easy to solve the linear equation system (4). Moreover, the complexity of computing the inverse matrix $\boldsymbol{A}$ is as high as $O(n^3)$. Hence, we need to combine the feature of the LS-SVM to find some more efficient methods to solve the large-scale linear system (4) or optimization problem (1).

3 Related works of the LS-SVM solution

This section will briefly introduce several mainstream algorithms for the fast training of the LS-SVM.

3.1 Conjugate gradient

The conjugate gradient (CG) is an important method for solving large-scale linear equations. But this method requires the coefficient matrix to be symmetric and positive definite. However, the matrix $\boldsymbol{A}$ in (4) is a non-positive matrix. Hence, the CG method cannot be used directly to solve the problem. Based on this, Suykens et al. [10] presented a linear system with the same solution as the linear system (4), that is

$$\begin{pmatrix} \boldsymbol{y}^T\boldsymbol{\Lambda}^{-1}\boldsymbol{y} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Lambda} \end{pmatrix}\begin{pmatrix} b \\ \boldsymbol{\alpha} + \boldsymbol{\Lambda}^{-1}\boldsymbol{y}b \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}^T\boldsymbol{\Lambda}^{-1}\boldsymbol{I} \\ \boldsymbol{I} \end{pmatrix}, \tag{5}$$

where $\boldsymbol{\Lambda} := \boldsymbol{K} + \gamma^{-1}\boldsymbol{I}(\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^T \succ 0)$. Hence, solving the linear equations (4) is transformed into the solving linear equations (5). This method makes full of use of the properties of the symmetric matrix $\boldsymbol{A}$. The detailed steps for training LS-SVM using the CG method are summarized in Algorithm 1.

Algorithm 1 CG for solving LS-SVM [10]

Input: training data $\boldsymbol{T}$, stopping criterion ϵ;
Output: dual variable $\boldsymbol{\alpha}^*$, bias term b^*;
1: Compute the matrix $\boldsymbol{\Lambda}$ according to the kernel matrix $\boldsymbol{K}$;
2: Solve the linear equations $\boldsymbol{\Lambda\eta} = \boldsymbol{y}$ and $\boldsymbol{\Lambda\vartheta} = \mathbf{1}$ using the CG method;
3: Get the variable μ according to $\mu = \boldsymbol{y}^T\boldsymbol{\eta}$;
4: Compute bias term $b^* = \boldsymbol{\eta}^T\mathbf{1}/\mu$ and dual variable $\boldsymbol{\alpha}^* = \boldsymbol{\vartheta} - b^*\boldsymbol{\eta}$;

Note that, the method of Suykens [10] needs to use the twice CG to solve the linear equations (5), which increase the amount of arithmetic. Chu [11] proposed a novel computation method based on the characteristics of the block matrix $\boldsymbol{K} + \gamma^{-1}\boldsymbol{I} = \begin{pmatrix} \boldsymbol{Q} & \boldsymbol{q} \\ \boldsymbol{q}^T & \boldsymbol{Q}_{nn} \end{pmatrix}$. Here, $\boldsymbol{Q} \in \mathbb{R}^{(n-1)\times(n-1)}$, $\boldsymbol{Q}_{nn} \in \mathbb{R}$ and $\boldsymbol{q} \in \mathbb{R}^{n-1}$. This method not only reduces the order of the linear equations (4) but also calls the CG once. This greatly reduces the amount of arithmetic. Specifically, the CG is first used to solve the linear system

$$\widetilde{\boldsymbol{Q}}\widetilde{\boldsymbol{\alpha}} = \widetilde{\boldsymbol{y}} - y_n\boldsymbol{I}_{n-1}. \tag{6}$$

Then, the optimal solution is obtained according to the $\boldsymbol{\alpha}^* = \begin{pmatrix} \widetilde{\boldsymbol{\alpha}}^* \\ -\boldsymbol{I}_{n-1}^T\widetilde{\boldsymbol{\alpha}}^* \end{pmatrix}$ and $b^* = y_n + \boldsymbol{Q}_{nn}(\boldsymbol{I}_{n-1}^T\widetilde{\boldsymbol{\alpha}}^*) - \boldsymbol{q}^T\widetilde{\boldsymbol{\alpha}}^*$, where $\widetilde{\boldsymbol{Q}} = \boldsymbol{Q} - \boldsymbol{I}_{n-1}\boldsymbol{q}^T - \boldsymbol{q}\boldsymbol{I}_{n-1}^T + \boldsymbol{Q}_{nn}\boldsymbol{I}_{n-1}\boldsymbol{I}_{n-1}^T$ and $\widetilde{\boldsymbol{y}} = (y_1, y_2, \cdots, y_{n-1})^T$. The detailed steps for training the LS-SVM using improved CG (ICG) method are summarized in Algorithm 2

optimization during each iteration. Compared with CG, SMO is simpler to implement and can deal with large-scale datasets. In some cases, the performance of SMO may be better than CG.

According to the Wolf duality theory, the dual problem of (1) is

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \mathcal{D}(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j[\widetilde{\boldsymbol{K}}]_{ij} - \sum_{i=1}^{n}\alpha_i y_i \\ &s.t. \quad \sum_{i=1}^{n}\alpha_i = 0, \end{aligned} \tag{7}$$

where $[\widetilde{\boldsymbol{K}}]_{ij} = [\boldsymbol{K}]_{ij} + \delta_{ij}/\gamma = k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \delta_{ij}/\gamma$ and $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. Obviously, dual problem (7) is a simple convex optimization problem with a equality constraint. The Lagrangian function of (7) is

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\alpha}, v) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j[\widetilde{\boldsymbol{K}}]_{ij} - \sum_{i=1}^{n}\alpha_i y_i + v\sum_{i=1}^{n}\alpha_i, \tag{8}$$

where v is Lagrangian multiplier. It follows from the KKT condition of (7) that

Algorithm 2 ICG for solving LS-SVM [11]

Input: training data $\boldsymbol{T}$, stopping criterion ϵ;
Output: dual variable $\boldsymbol{\alpha}^*$, bias term b^*;
1: Compute the matrix $\widetilde{\boldsymbol{Q}}$;
2: Solve the linear equations (6) using the CG method;
3: Compute dual variable; $\boldsymbol{\alpha}^* = \begin{pmatrix} \widetilde{\alpha}^* \\ -\mathbf{1}_{n-1}^T\widetilde{\boldsymbol{\alpha}}^* \end{pmatrix}$;
4: Compute the bias term $b^* = y_n + \boldsymbol{Q}_{nn}(\mathbf{1}_{n-1}^T\widetilde{\boldsymbol{\alpha}}^*) - \boldsymbol{q}^T\widetilde{\boldsymbol{\alpha}}^*$;

3.2 First-order SMO

The CG method starts from the primal problem to obtain the optimal solution of problem (1). However, solving the dual problem of (1) can also get the optimal solution according to the duality theory. The sequential minimal optimization (SMO) is an algorithm designed for the dual problem of (1). To satisfy the constraints of the dual problem, the SMO algorithm only selects two variables for

$$\nabla_{\alpha_i}\mathcal{L}_{\mathcal{D}}(\boldsymbol{\alpha}, v) = \nabla_{\alpha_i}\mathcal{D}(\boldsymbol{\alpha}) + v = 0, \forall i, \tag{9}$$

where $\nabla_{\alpha_i}\mathcal{D}(\boldsymbol{\alpha}) = \sum_{j=1}^{n}\alpha_j[\widetilde{\boldsymbol{K}}]_{ij} - y_i = G_i(\boldsymbol{\alpha}), \forall i$. Note that, when $\max_i(G_i(\boldsymbol{\alpha})) = \min_i(G_i(\boldsymbol{\alpha}))$ holds and $\sum_{i=1}^{n}\boldsymbol{\alpha}_i = 0$, then $\boldsymbol{\alpha}$ is the optimal solution of the dual problem (7). In actual computation, the strict KKT conditions are generally not used, but a small positive number ϵ is set in advance so that

$$\left\{\max_i(G_i(\boldsymbol{\alpha})) - \min_i(G_i(\boldsymbol{\alpha}))\right\} \leq \epsilon \tag{10}$$

is established. This method has also been adopted by the LIBSVM [31] and SVM Light [32]. Let (α_i^k, α_j^k) be the variable pair selected in the k-th iteration. Since the equation constraint $\sum_{i=1}^n \alpha_i = 0$ must be satisfied, the update of (α_i^k, α_j^k) needs to satisfy

$$\begin{aligned}
\alpha_i^{k+1} &\Longleftarrow \alpha_i^k + \Delta\boldsymbol{\alpha}^k \\
\alpha_j^{k+1} &\Longleftarrow \alpha_j^k - \Delta\boldsymbol{\alpha}^k \\
\alpha_\ell^{k+1} &\Longleftarrow \alpha_\ell^k, \forall \ell \neq i, j.
\end{aligned} \tag{11}$$

$\Delta\boldsymbol{\alpha}^k$ is the amount of variation for α_i^k and α_j^k. If the functional gain is denoted by $f_G(\Delta\boldsymbol{\alpha}^k)$, we have

$$\begin{aligned}
f_G(\Delta\boldsymbol{\alpha}^k) =& \mathcal{D}(\boldsymbol{\alpha}^k) - \mathcal{D}(\boldsymbol{\alpha}^{k+1}) \\
=& \frac{1}{2}(\boldsymbol{\alpha}^k)^T \widetilde{\boldsymbol{K}} \boldsymbol{\alpha}^k - \boldsymbol{y}^T \boldsymbol{\alpha}^k - \frac{1}{2}(\boldsymbol{\alpha}^{k+1})^T \widetilde{\boldsymbol{K}} \boldsymbol{\alpha}^{k+1} + y^T \boldsymbol{\alpha}^{k+1} \\
=& -\frac{1}{2}\begin{pmatrix} -\Delta\boldsymbol{\alpha}^k \\ \Delta\boldsymbol{\alpha}^k \end{pmatrix}^T \begin{pmatrix} \left[\widetilde{\boldsymbol{K}}\right]_{ii} & \left[\widetilde{\boldsymbol{K}}\right]_{ij} \\ \left[\widetilde{\boldsymbol{K}}\right]_{ji} & \left[\widetilde{\boldsymbol{K}}\right]_{jj} \end{pmatrix} \begin{pmatrix} -\Delta\boldsymbol{\alpha}^k \\ \Delta\boldsymbol{\alpha}^k \end{pmatrix} \\
&+ \begin{pmatrix} G_i(\boldsymbol{\alpha}^k) \\ G_j(\boldsymbol{\alpha}^k) \end{pmatrix}^T \begin{pmatrix} -\Delta\boldsymbol{\alpha}^k \\ \Delta\boldsymbol{\alpha}^k \end{pmatrix}.
\end{aligned} \tag{12}$$

f_G is the quadratic function of $\Delta\boldsymbol{\alpha}^k$. Let $f_G'(\Delta\boldsymbol{\alpha}^k) = 0$, then we have

$$\Delta\boldsymbol{\alpha}^k = \frac{G_j(\boldsymbol{\alpha}^k) - G_i(\boldsymbol{\alpha}^k)}{\left[\widetilde{\boldsymbol{K}}\right]_{ii} + \left[\widetilde{\boldsymbol{K}}\right]_{jj} - \left[\widetilde{\boldsymbol{K}}\right]_{ij} - \left[\widetilde{\boldsymbol{K}}\right]_{ji}}. \tag{13}$$

Substituting $\Delta\boldsymbol{\alpha}^k$ into (12) yields the functional gain

$$f_G(\Delta\boldsymbol{\alpha}^k) = \frac{\left(G_j(\boldsymbol{\alpha}^k) - G_i(\boldsymbol{\alpha}^k)\right)^2}{2\left(\left[\widetilde{\boldsymbol{K}}\right]_{ii} + \left[\widetilde{\boldsymbol{K}}\right]_{jj} - \left[\widetilde{\boldsymbol{K}}\right]_{ij} - \left[\widetilde{\boldsymbol{K}}\right]_{ji}\right)}. \tag{14}$$

Obviously, working set (i, j) needs to be selected to maximize $f_G(\Delta\boldsymbol{\alpha}^k)$, namely

$$(i,j) = \mathop{\arg\min}_{m,\ell}\left\{\frac{\left(G_l(\boldsymbol{\alpha}^k) - G_m(\boldsymbol{\alpha}^k)\right)^2}{2\left(\left[\widetilde{\boldsymbol{K}}\right]_{mm} + \left[\widetilde{\boldsymbol{K}}\right]_{ll} - \left[\widetilde{\boldsymbol{K}}\right]_{ml} - \left[\widetilde{\boldsymbol{K}}\right]_{lm}\right)}\right\} \tag{15}$$

In the first-order SMO algorithm, Keerthi [14] neglected the denominator of $f_G(\Delta\boldsymbol{\alpha}^k)$ and only consider the numerator to reach the maximum, namely

$$(i,j) = \mathop{\arg\min}_{m,\ell}\left\{\left(G_m(\boldsymbol{\alpha}^k) - G_\ell(\boldsymbol{\alpha}^k)\right)^2\right\}. \tag{16}$$

Obviously, the selecting method of working set (i, j) is equivalent to

$$i = \mathop{\arg\min}_{\ell} G_\ell(\boldsymbol{\alpha}^k),\ j = \mathop{\arg\max}_{m} G_m(\boldsymbol{\alpha}^k). \tag{17}$$

At this time, working set (i, j) is also called the MVP. The first-order SMO proposed by Keerthi [14] uses the dual gap as the stopping condition. However, we uniformly use (10) as the stopping condition. The detailed computation process of the first-order SMO is summarized in Algorithm 3. Here, $\boldsymbol{G}(\boldsymbol{\alpha}^k) = \left(G_1(\boldsymbol{\alpha}^k), G_2(\boldsymbol{\alpha}^k), \cdots, G_n(\boldsymbol{\alpha}^k)\right)^T$ denotes the gradient of $\mathcal{D}(\boldsymbol{\alpha})$ at the k-th iteration.

Algorithm 3 First order SMO [14]

Input: training data $\boldsymbol{T}$, stopping criterion ϵ;
Output: dual variable $\boldsymbol{\alpha}^*$;
1: initialize $\boldsymbol{\alpha}^0 = \boldsymbol{0}$, $\boldsymbol{G}(\boldsymbol{\alpha}^0) = -\boldsymbol{y}$ and $k = 0$;
2: **while** Stop condition (10) not met **do**
3: Select the working (i, j) according to (17);
4: Compute $\Delta\boldsymbol{\alpha}^k$ with (13);
5: Updata dual variable $\boldsymbol{\alpha}^{k+1}$ with (11);
6: Updata gradient $\boldsymbol{G}(\boldsymbol{\alpha}^k)$;
7: let $k \leftarrow k + 1$ and go back to step 2;
8: **end while**

3.3 Second-order SMO

In LIBSVM [31], the SVMs model is trained using the second-order version of the SMO algorithm. Generally, the efficiency of the second-order SMO algorithm is higher than that of the first-order SMO algorithm [15]. Hence, López [16] extended the second-order SMO algorithm of the SVMs to the LS-SVM to improve the training efficiency of the LS-SVM.

Since the first-order SMO algorithm ignores the denominator of $f_G(\Delta\boldsymbol{\alpha}^k)$, the working set (i, j) selected with

the MVP may not maximize $f_G(\Delta\boldsymbol{\alpha}^k)$. Based on this, the second-order SMO algorithm of LS-SVM takes the denominator of $f_G(\Delta\boldsymbol{\alpha}^k)$ into consideration when selecting the working set (i, j). However, note that, in order to find the optimal working set (i, j), it is necessary to traverse (15) to find the working set (i, j) corresponding to the maximum value, and the complexity is $O(n^2)$. When the sample size n is large, this is unacceptable. Fan [15] and López [16] used a compromise method. This method first uses the first-order SMO to find the coordinate i and then traverses (15) to find the coordinate j corresponding to the maximum value, namely

$$i = \arg\min_{\ell} G_\ell(\boldsymbol{\alpha}^k),$$
$$j = \arg\max_{\ell \neq i} \left\{ \frac{\left(G_\ell(\boldsymbol{\alpha}^k) - G_i(\boldsymbol{\alpha}^k)\right)^2}{2\left(\left[\widetilde{\boldsymbol{K}}\right]_{ii} + \left[\widetilde{\boldsymbol{K}}\right]_{\ell\ell} - \left[\widetilde{\boldsymbol{K}}\right]_{i\ell} - \left[\widetilde{\boldsymbol{K}}\right]_{\ell i}\right)} \right\}. \tag{18}$$

Although the second-order SMO algorithm increases some kernel operations, the functional gain of the dual function is greater than that of the first-order SMO algorithm in each iteration. Fan [15] and López's [16] numerical experiments have shown that the efficiency of the second-order SMO is generally higher than that of the first-order SMO. But when the penalty parameter $\gamma \approx 0$, the kernel parameter $\sigma \approx 0$ and $\sigma \gg 0$ of the RBF, the second-order SMO is almost equivalent to the first-order SMO [16]. Replacing the working set selection method of step 3 in Algorithm 3 with (18) is the computation process of the second-order SMO algorithm.

3.4 Functional gain SMO

The coordinate index corresponding to the minimum value of the gradient $\boldsymbol{G}(\boldsymbol{\alpha})$ is the selected i in the second-order SMO algorithm. There may be a problem with this selection method, namely, $G_i(\boldsymbol{\alpha})$ may not be the most obvious gradient component that violates the KKT condition, namely, there may be a $\hat{i}$ that makes $|G_i(\boldsymbol{\alpha})| \leq G_{\hat{i}}(\boldsymbol{\alpha})$. Hence, Bo [18] used functional gain to select the working set (i, j). Specifically, the selection of index i is the coordinate index corresponding to the maximum value of $|\boldsymbol{G}(\boldsymbol{\alpha})|$, and the selection method of index j is consistent with the second-order SMO, namely

$$i = \arg\max_{\ell} \left|G_\ell(\boldsymbol{\alpha}^k)\right|,$$
$$j = \arg\max_{\ell \neq i} \left\{ \frac{\left(G_\ell(\boldsymbol{\alpha}^k) - G_i(\boldsymbol{\alpha}^k)\right)^2}{2\left(\left[\widetilde{\boldsymbol{K}}\right]_{ii} + \left[\widetilde{\boldsymbol{K}}\right]_{\ell\ell} - \left[\widetilde{\boldsymbol{K}}\right]_{i\ell} - \left[\widetilde{\boldsymbol{K}}\right]_{\ell i}\right)} \right\}. \tag{19}$$

This selection method was called FGWSS (functional gain working selection strategy). Obviously, the method of selecting the working set (i, j) by the second-order SMO is only a special case of the FGWSS method. Because $i = \arg\min_\ell\left(G_\ell(\alpha^k)\right)$ or $i = \arg\min_\ell\left(-G_\ell(\alpha^k)\right) = \arg\max_\ell\left(G_\ell(\alpha^k)\right)$. In the case of the same gradient $\boldsymbol{G}(\boldsymbol{\alpha})$, using the working set (i, j) selected by FGWSS can always ensure that the variation in the dual function is greater than or equal to the MVP method. However, second-order SMO may not have this property. The computation process of the functional gain SMO algorithm is not significantly different from that of the second-order SMO algorithm. It only needs to replace the working set selection method in Algorithm 3 with (19), and the rest of the computation steps are exactly the same.

4 The proposed conjugate functional gain SMO

The only difference between the three SMO algorithms described above is the way in which the working set is selected. The unified iteration format of the SMO algorithm is $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho_k \boldsymbol{h}_{ij}^k$, where $\boldsymbol{h}_{ij}^k = \boldsymbol{e}_i^k - \boldsymbol{e}_j^k$ is the direction of the k-th iteration of the SMO algorithm, ρ_k is the step length parameter, and (i, j) is the working set. Next, we will use a conjugate direction $\boldsymbol{s}_k$ to replace the direction $\boldsymbol{h}_{ij}^k$. Consider iterative format

$$\begin{aligned} \boldsymbol{\alpha}^{k+1} &= \boldsymbol{\alpha}^k + \rho_k \boldsymbol{s}_k \\ \boldsymbol{s}_k &= \boldsymbol{h}_{ij}^k + r_k \boldsymbol{s}_{k-1}, \end{aligned} \tag{20}$$

where the direction $\boldsymbol{h}_{ij}^k$ is determined by FGWSS, and r_k is the conjugate parameter. Because $\boldsymbol{\alpha}^{k+1}$ must satisfy the equality constraint, namely

$$\sum_{i=1}^{n} \alpha_i^{k+1} = \sum_{i=1}^{n} \alpha_i^k + \rho_k \sum_{i=1}^{n} s_k^i = \sum_{i=1}^{n} \alpha_i^k + \rho_k r_k \sum_{i=1}^{n} s_{k-1}^i = 0. \tag{21}$$

Hence, if $\sum_{i=1}^{n} s_{k-1}^i = 0$ holds, then (21) naturally holds. By the line search criterion, let $\varphi(\rho_k) = \mathcal{D}(\boldsymbol{\alpha}^k + \rho_k \boldsymbol{s}_k)$, then according to $\varphi'(\rho_k) = \rho_k \boldsymbol{s}_k^T \widetilde{\boldsymbol{K}} \boldsymbol{s}_k + \boldsymbol{s}_k^T \boldsymbol{G}(\boldsymbol{\alpha}^k) = 0$, we can obtain the optimal step length parameter

$$\begin{aligned} \rho_k^* &= -\frac{\boldsymbol{s}_k^T \boldsymbol{G}(\boldsymbol{\alpha}^k)}{\boldsymbol{s}_k^T \widetilde{\boldsymbol{K}} \boldsymbol{s}_k} = \frac{-\left(\boldsymbol{h}_{ij}^k + r_k \boldsymbol{s}_{k-1}\right)^T \boldsymbol{G}(\boldsymbol{\alpha}^k)}{\boldsymbol{s}_k^T \widetilde{\boldsymbol{K}} \boldsymbol{s}_k} \\ &= \frac{-\boldsymbol{G}(\boldsymbol{\alpha}^k)^T \boldsymbol{h}_{ij}^k - r_k \boldsymbol{s}_{k-1}^T \boldsymbol{G}(\boldsymbol{\alpha}^k)}{\boldsymbol{s}_k^T \widetilde{\boldsymbol{K}} \boldsymbol{s}_k} \end{aligned} \tag{22}$$

Note that, $\boldsymbol{s}_k^T \boldsymbol{G}(\boldsymbol{\alpha}^{k+1}) = 0$. Hence, (22) can be further simplified as $\rho_k^* = -\boldsymbol{G}(\boldsymbol{\alpha}^k)^T \boldsymbol{h}_{ij}^k / \boldsymbol{s}_k^T \widetilde{\boldsymbol{K}} \boldsymbol{s}_k$. Since the inner product of the direction $\boldsymbol{s}_k$ and the gradient $\boldsymbol{G}(\boldsymbol{\alpha}^k)$ satisfies

$s_k^T G(\alpha^k) = G(\alpha^k)^T h_{ij}^k < 0$, the direction s_k is a descent direction. At this time, the functional gain is

$$\begin{aligned} f_G(\Delta\alpha^k) &= \mathcal{D}(\alpha^k) - \mathcal{D}(\alpha^{k+1}) \\ &= -\frac{1}{2}\rho^2 s_k^T \tilde{K} s_k - \rho s_k^T \left(\tilde{K}\alpha^k - y\right). \end{aligned} \tag{23}$$

Substitute $\rho_k^* = -G(\alpha^k)^T h_{ij}^k / s_k^T \tilde{K} s_k$ and $s_{k-1}^T G(\alpha^k) = 0$ into (23) to get

$$f_G(\Delta\alpha^k) = \mathcal{D}(\alpha^k) - \mathcal{D}(\alpha^{k+1}) = \frac{1}{2}\frac{\left(G(\alpha^k)^T h_{ij}^k\right)^2}{s_k^T \tilde{K} s_k}, \tag{24}$$

where

$$\psi(r) = s_k^T \tilde{K} s_k = \left(h_{ij}^k + r s_{k-1}\right)^T \tilde{K} \left(h_{ij}^k + r s_{k-1}\right).$$

Because there is a parameter r in the denominator $\psi(r)$, $f_G(\Delta\alpha^k)$ can be further maximized. Let $\psi'(r) = 2\left(s_k^T \tilde{K} h_{ij}^k + r s_{k-1} \tilde{K} s_{k-1}\right) = 0$, we have

$$r^* = -\frac{s_{k-1}^T \tilde{K} h_{ij}^k}{s_{k-1}^T \tilde{K} s_{k-1}}. \tag{25}$$

Notice that substituting (25) into (20) yields

$$\begin{aligned} s_{k-1}^T \tilde{K} s_{k-1} &= s_{k-1}^T \tilde{K} h_{ij}^k + r_k s_{k-1}^T \tilde{K} s_{k-1} \\ &= s_{k-1}^T \tilde{K} h_{ij}^k - \frac{s_{k-1}^T \tilde{K} h_{ij}^k}{s_{k-1}^T \tilde{K} s_{k-1}} s_{k-1}^T \tilde{K} s_{k-1} = 0. \end{aligned} \tag{26}$$

Because $\tilde{K}$ is symmetric and positive definite, s_k is conjugate with s_{k-1}. The above computation process is the conjugate functional gain SMO (CFGSMO) algorithm of LS-SVM. Algorithm 4 is the CFGSMO, and Fig. 1 shows its flowchart.

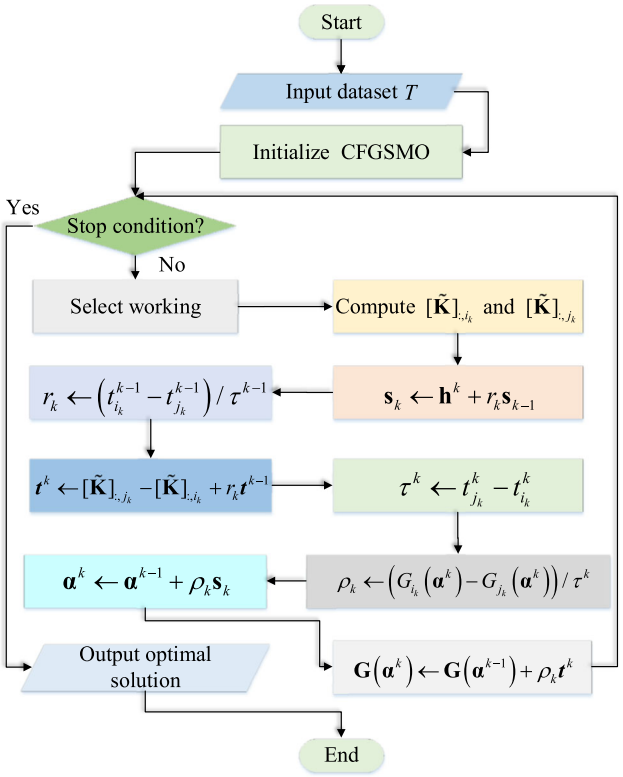

Fig. 1 The flowchart of CFGSMO for solving LS-SVM

CFGSMO is an extension of the conjugate SMO (CSMO) [28] for solving SVMs. The dual problem of SVMs has a box constraint $0 \le \alpha_i \le C, \forall i$, which indicates that the dual decision variable α is confined within an interval. Whether it is the plain SMO or the CSMO, when α exceeds the box constraint, it is necessary to strictly limit α within the interval determined by the box constraint

Algorithm 4 Conjugate functional gain SMO

Input: training data T, stopping criterion ϵ;
Output: dual variable α^*;

initialize $\alpha^0 = 0$, $G(\alpha^0) = 0$, $s_0 = t_0 = 0$, $\tau^0 = 1$ and $k = 0$;
while Stop condition (10) not met **do**
  $k \leftarrow k + 1$;
  Select working set (i_k, j_k) according to (19);
  Compute the kernel matrix elements $[\tilde{K}]_{:,i_k}$ and $[\tilde{K}]_{:,j_k}$;
  $r_k \leftarrow \left(t_{i_k}^{k-1} - t_{j_k}^{k-1}\right)/\tau^{k-1}$;
  $s_k \leftarrow h^k + r_k s_{k-1}$;
  $t^k \leftarrow [\tilde{K}]_{:,j_k} - [\tilde{K}]_{:,i_k} + r_k t^{k-1}$;
  $\tau^k \leftarrow t_{j_k}^k - t_{i_k}^k$;
  Compute $\rho_k \leftarrow \left(G_{i_k}(\alpha^k) - G_{j_k}(\alpha^k)\right)/\tau^k$;
  Updata dual variable $\alpha^k \leftarrow \alpha^{k-1} + \rho_k s_k$;
  Updata gradient $G(\alpha^k) \leftarrow G(\alpha^{k-1}) + \rho_k t^k$;
end while

through the clipping operation. Not only that, when $\boldsymbol{\alpha}$ are clipped, the CSMO also needs to restart the entire algorithm with the plain SMO. However, the dual problem of LS-SVM does not contain box constraints. This means that the CFGSMO is not affected by the boundary caused by the box constraint, so it does not need to clip the dual decision variables $\boldsymbol{\alpha}$, and it does not have the restart step of the CSMO. Hence, the CFGSMO algorithm will be simpler.

5 Convergence

The first-order SMO, second-order SMO, and FGSMO algorithms are proven to converge. Naturally, whether the CFGSMO algorithm used to train the LS-SVM converges to an optimal solution or not should also be discussed. Hence, this section briefly discusses the convergence of CFGSMO algorithm.

5.1 Asymptotic

Next, we will illustrate this problem through several lemmas and a theorem. Lemma 5.1 is only an extension of the conclusion in [28].

Lemma 5.1 *In the process of using the CFGSMO algorithm to train the LS-SVM model, the functional gain $f_G(\Delta\boldsymbol{\alpha})_{CF}$ of the CFGSMO is always greater than the functional gain $f_G(\Delta\boldsymbol{\alpha})_F$ of the FGSMO algorithm.*

Proof Suppose that in the k-th iteration, the working set selected by the FGSMO algorithm is (i, j), so $\left(\alpha_i^k, \alpha_j^k\right)$ is the variable selected in the k-th iteration. At this time, the functional gain is

$$f_G(\Delta\boldsymbol{\alpha})_F = \frac{\left(G_j(\alpha^k) - G_i(\alpha^k)\right)^2}{2\left(\left[\widetilde{\boldsymbol{K}}\right]_{ii} + \left[\widetilde{\boldsymbol{K}}\right]_{jj} - \left[\widetilde{\boldsymbol{K}}\right]_{ij} - \left[\widetilde{\boldsymbol{K}}\right]_{ji}\right)} = \frac{1}{2}\frac{\left(\boldsymbol{G}(\boldsymbol{\alpha}^k)^T \boldsymbol{h}_{ij}^k\right)^2}{\left(\boldsymbol{h}_{ij}^k\right)^T \widetilde{\boldsymbol{K}} h_{ij}^k}, \tag{27}$$

where $\boldsymbol{h}_{ij}^k$ is a descent direction of FGSMO algorithm in the k-th iteration. And because of

$$\begin{aligned} \boldsymbol{s}_k^T \widetilde{\boldsymbol{K}} \boldsymbol{s}_k &= \left(\boldsymbol{h}_{ij}^k\right)^T \widetilde{\boldsymbol{K}} \boldsymbol{h}_{ij}^k - \frac{\left(\boldsymbol{s}_{k-1}^T \widetilde{\boldsymbol{K}} \boldsymbol{h}_{ij}^k\right)^2}{\boldsymbol{s}_{k-1}^T \widetilde{\boldsymbol{K}} \boldsymbol{s}_{k-1}} \\ &= \left\|\boldsymbol{h}_{ij}^k\right\|_{\widetilde{\boldsymbol{K}}}^2 \left(1 - \frac{\left(\boldsymbol{s}_{k-1}^T \widetilde{\boldsymbol{K}} \boldsymbol{h}_{ij}^k\right)^2}{\left\|\boldsymbol{h}_{ij}^k\right\|_{\widetilde{\boldsymbol{K}}}^2 \|\boldsymbol{s}_{k-1}\|_{\widetilde{\boldsymbol{K}}}^2}\right), \end{aligned} \tag{28}$$

where $\left\|\boldsymbol{h}_{ij}^k\right\|_{\widetilde{\boldsymbol{K}}}^2 = \left(\boldsymbol{h}_{ij}^k\right)^T \widetilde{\boldsymbol{K}} \boldsymbol{h}_{ij}^k$. From (23), the functional gain of the CFGSMO algorithm can be further expressed as

$$\begin{aligned} f_G(\Delta\boldsymbol{\alpha}^k)_{CF} &= \frac{1}{2}\frac{\left(\boldsymbol{G}(\boldsymbol{\alpha}^k)^T \boldsymbol{h}_{ij}^k\right)^2}{\boldsymbol{s}_k^T \widetilde{\boldsymbol{K}} \boldsymbol{s}_k} \\ &= \frac{1}{2}\frac{\left(\boldsymbol{G}(\boldsymbol{\alpha}^k)^T \boldsymbol{h}_{ij}^k\right)^2}{\left\|\boldsymbol{h}_{ij}^k\right\|_{\widetilde{\boldsymbol{K}}}^2 \left(1 - \frac{\left(\boldsymbol{s}_{k-1}^T \widetilde{\boldsymbol{K}} \boldsymbol{h}_{ij}^k\right)^2}{\|\boldsymbol{h}_{ij}^k\|_{\widetilde{\boldsymbol{K}}}^2 \|\boldsymbol{s}_{k-1}\|_{\widetilde{\boldsymbol{K}}}^2}\right)}. \end{aligned} \tag{29}$$

Since $\boldsymbol{s}_{k-1}^T \widetilde{\boldsymbol{K}} \boldsymbol{h}_{ij}^k \le \left\|\boldsymbol{h}_{ij}^k\right\|_{\widetilde{\boldsymbol{K}}} \|\boldsymbol{s}_{k-1}\|_{\widetilde{\boldsymbol{K}}}$ holds, we can therefore obtain $f_G(\Delta\boldsymbol{\alpha})_F \le f_G(\Delta\boldsymbol{\alpha})_{CF}$. □

Lemma 5.2 *For the CFGSMO algorithm, when $k \ge 1$, the functional gain of any two adjacent iterations of the dual function satisfies*

$$f_G(\Delta\boldsymbol{\alpha}^k)_{CF} \ge \frac{\left\|\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k+1}\right\|^2}{2\gamma}. \tag{30}$$

Proof Suppose that the working set selected by the MVP is (i, j), and the working set selected by the FGWSS is (i_F, j_F). By the Lemma 5.1, it follows that

$$\begin{aligned} f_G(\Delta\boldsymbol{\alpha})_F &= \frac{\left(G_j(\boldsymbol{\alpha}^k) - G_i(\boldsymbol{\alpha}^k)\right)^2}{2\left(\left[\widetilde{\boldsymbol{K}}\right]_{ii} + \left[\widetilde{\boldsymbol{K}}\right]_{jj} - \left[\widetilde{\boldsymbol{K}}\right]_{ij} - \left[\widetilde{\boldsymbol{K}}\right]_{ji}\right)} = \frac{1}{2}\frac{\left(\boldsymbol{G}(\boldsymbol{\alpha}^k)^T \boldsymbol{h}_{ij}^k\right)^2}{\left(\boldsymbol{h}_{ij}^k\right)^T \widetilde{\boldsymbol{K}} h_{ij}^k} \le \frac{\left(G_{j_F}(\boldsymbol{\alpha}^k) - G_{i_F}(\boldsymbol{\alpha}^k)\right)^2}{2\left(\left[\widetilde{\boldsymbol{K}}\right]_{i_F i_F} + \left[\widetilde{\boldsymbol{K}}\right]_{j_F j_F} - \left[\widetilde{\boldsymbol{K}}\right]_{i_F j_F} - \left[\widetilde{\boldsymbol{K}}\right]_{j_F i_F}\right)} \\ &= \frac{1}{2}\frac{\left(\boldsymbol{G}(\boldsymbol{\alpha}^k)^T \boldsymbol{h}_{i_F j_F}^k\right)^2}{\left(\boldsymbol{h}_{i_F j_F}^k\right)^T \widetilde{\boldsymbol{K}} h_{i_F j_F}^k} \le \frac{1}{2}\frac{\left(\boldsymbol{G}(\boldsymbol{\alpha}^k)^T \boldsymbol{h}_{i_F j_F}^k\right)^2}{\boldsymbol{s}_k^T \widetilde{\boldsymbol{K}} \boldsymbol{s}_k} = f_G(\boldsymbol{\alpha}^k)_{CF}. \end{aligned} \tag{31}$$

As $f_G(\Delta\boldsymbol{\alpha}^k)_F \geq \frac{\left\|\boldsymbol{\alpha}^{k+1}-\boldsymbol{\alpha}^k\right\|^2}{2\gamma}$ holds, we can therefore obtain

$$f_G(\Delta\boldsymbol{\alpha}^k)_{CF} = \frac{1}{2}\frac{\left(\boldsymbol{G}(\boldsymbol{\alpha}^k)^T\boldsymbol{h}^k_{i_F j_F}\right)^2}{\boldsymbol{s}_k^T\widetilde{\boldsymbol{K}}\boldsymbol{s}_k} \geq \frac{\left\|\boldsymbol{\alpha}^{k+1}-\boldsymbol{\alpha}^k\right\|^2}{2\gamma}. \tag{32}$$

□

Lemma 5.3 *The dual function $\mathcal{D}(\boldsymbol{\alpha})$ has a lower bound $-2\sum y_i^2/\sqrt{\lambda_{\min}(\widetilde{\boldsymbol{K}})}$, where $\lambda_{\min}(\widetilde{\boldsymbol{K}})>0$ denotes the smallest eigenvalue of the matrix $\widetilde{\boldsymbol{K}}$.*

Proof Obviously, the dual function satisfies inequality

$$\mathcal{D}(\alpha) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\left[\widetilde{\boldsymbol{K}}\right]_{ij} - \sum_{i=1}^{n}\alpha_i y_i \geq -\sum_{i=1}^{n}\alpha_i y_i. \tag{33}$$

By the Cauchy inequality, we can get

$$\sum_{i=1}^{n} y_i\alpha_i \leq \sqrt{\sum_{i=1}^{n} y_i^2}\sqrt{\sum_{i=1}^{n}\alpha_i^2}. \tag{34}$$

Because $\lambda_{\min}(\widetilde{\boldsymbol{K}})\boldsymbol{\alpha}^T\boldsymbol{\alpha} \leq \boldsymbol{\alpha}^T\widetilde{\boldsymbol{K}}\boldsymbol{\alpha}$ holds, so we have $\sqrt{\sum_i \alpha_i^2} = \sqrt{\boldsymbol{\alpha}^T\boldsymbol{\alpha}} \leq \sqrt{\boldsymbol{\alpha}^T\widetilde{\boldsymbol{K}}\boldsymbol{\alpha}/\lambda_{\min}(\widetilde{\boldsymbol{K}})}$ and $\sum_i \alpha_i y_i \leq \sqrt{\sum_i y_i^2}\sqrt{\boldsymbol{\alpha}^T\widetilde{\boldsymbol{K}}\boldsymbol{\alpha}/\lambda_{\min}(\widetilde{\boldsymbol{K}})}$. Note that, $\boldsymbol{0}$ is a feasible solution of dual problem (7), it follows that

$$\frac{1}{2}\boldsymbol{\alpha}^T\widetilde{\boldsymbol{K}}\boldsymbol{\alpha} \leq \sum_i \alpha_i y_i \leq \sqrt{\sum_i y_i^2}\sqrt{\boldsymbol{\alpha}^T\widetilde{\boldsymbol{K}}\boldsymbol{\alpha}/\lambda_{\min}(\widetilde{\boldsymbol{K}})}. \tag{35}$$

According to (35), $\sqrt{\boldsymbol{\alpha}^T\widetilde{\boldsymbol{K}}\boldsymbol{\alpha}} \leq 2\sqrt{\sum_i y_i^2/\lambda_{\min}(\widetilde{\boldsymbol{K}})}$ can be derived. Hence, $\sum_i \alpha_i y_i \leq 2\sum_i y_i^2/\lambda_{\min}(\widetilde{\boldsymbol{K}})$. To sum up, $\mathcal{D}(\boldsymbol{\alpha}) \geq -2\sum_i y_i^2/\sqrt{\lambda_{\min}(\widetilde{\boldsymbol{K}})}$ is true, as desired. □

Theorem 5.1 *The sequence $\{\boldsymbol{\alpha}^k\}$ generated by the CFGSMO algorithm converges to the global optimal solution $\boldsymbol{\alpha}^*$ of the dual problem (7).*

Proof It is known from Lemma 5.2 that the dual function $\mathcal{D}(\boldsymbol{\alpha})$ is strictly decreasing in the whole iteration process, because the change $\boldsymbol{G}(\boldsymbol{\alpha})^T\boldsymbol{h}_{i_F j_F} \neq 0$. According to the Lemma 5.3, $f_G(\Delta\boldsymbol{\alpha}^k)_{CF}$ converges to 0. It follows that the sequence $\{\boldsymbol{\alpha}^{k+1}-\boldsymbol{\alpha}^k\}$ also converges to $\boldsymbol{0}$. Because the matrix $\widetilde{\boldsymbol{K}}$ is a symmetric positive definite matrix, the dual function $\mathcal{D}(\boldsymbol{\alpha})$ is strictly convex, there is a unique global optimal solution $\boldsymbol{\alpha}^*$, and $\{\boldsymbol{\alpha}^k\} \subset \{\boldsymbol{\alpha}^t \mid \mathcal{D}(\boldsymbol{\alpha}^t) \leq \mathcal{D}(\boldsymbol{\alpha}^0)\}$ is a compact set. Hence, $\{\boldsymbol{\alpha}^k\}$ has a subsequence $\{\boldsymbol{\alpha}^{k_s}\}$ with $\hat{\boldsymbol{\alpha}}$ as the limit point. Note that, dual variable $\boldsymbol{\alpha} \in \mathbb{R}^n$. So there are at most n^2 options for working set (i_F, j_F). Hence, there must be an infinite number of times a working set is selected. Without loss of generality, suppose the working set is (i_F, j_F). Consider the limit

$$\begin{aligned} G_{i_F}(\hat{\boldsymbol{\alpha}}) - G_{j_F}(\hat{\boldsymbol{\alpha}}) &= \lim_{k_s\to\infty}\left(G_{i_F}\left(\boldsymbol{\alpha}^{k_s}\right) - G_{j_F}\left(\boldsymbol{\alpha}^{k_s}\right)\right) \\ &= \lim_{k_s\to\infty}\left(A(k_s) + B(k_s) + C(k_s)\right), \end{aligned} \tag{36}$$

where $A(k_s) = G_{i_F}\left(\boldsymbol{\alpha}^{k_s}\right) - G_{i_F}\left(\boldsymbol{\alpha}^{k_s+1}\right)$, $B(k_s) = G_{i_F}\left(\boldsymbol{\alpha}^{k_s+1}\right) - G_{j_F}\left(\boldsymbol{\alpha}^{k_s+1}\right)$ and $C(k_s) = G_{j_F}\left(\boldsymbol{\alpha}^{k_s+1}\right) - G_{j_F}\left(\boldsymbol{\alpha}^{k_s}\right)$. Because $\{\boldsymbol{\alpha}^{k+1}-\boldsymbol{\alpha}^k\}$ converges to $\boldsymbol{0}$, both $A(k_s)$ and $C(k_s)$ converge to $\boldsymbol{0}$ when $k_s \to \infty$. Obviously, $B(k_s) = 0$ always holds. It follows that $G_{i_F}(\hat{\boldsymbol{\alpha}}) - G_{j_F}(\hat{\boldsymbol{\alpha}}) = 0$. Recalling the selection method of working set (i_F, j_F), we consider another limit

$$\left(G_i(\hat{\boldsymbol{\alpha}}) - G_j(\hat{\boldsymbol{\alpha}})\right)^2 = \left(\lim_{k_s\to\infty}\left(G_i\left(\boldsymbol{\alpha}^{k_s}\right) - G_j\left(\boldsymbol{\alpha}^{k_s}\right)\right)\right)^2, \forall i,j. \tag{37}$$

Furthermore, by the conclusion of Lemma 5.1 and the proof process of Lemma 5.2, we have

$$\begin{aligned} \left(G_i(\hat{\boldsymbol{\alpha}}) - G_j(\hat{\boldsymbol{\alpha}})\right)^2 &= \left(\lim_{k_s\to\infty}\left(G_i\left(\boldsymbol{\alpha}^{k_s}\right) - G_j\left(\boldsymbol{\alpha}^{k_s}\right)\right)\right)^2 \\ &\leq \frac{\boldsymbol{h}_{ij}^T\widetilde{\boldsymbol{K}}\boldsymbol{h}_{ij}}{\boldsymbol{s}_k^T\widetilde{\boldsymbol{K}}\boldsymbol{s}_k}\left(\lim_{k_s\to\infty}\left(G_{i_F}\left(\boldsymbol{\alpha}^{k_s}\right) - G_{j_F}\left(\boldsymbol{\alpha}^{k_s}\right)\right)\right)^2 \\ &= \frac{\boldsymbol{h}_{ij}^T\widetilde{\boldsymbol{K}}\boldsymbol{h}_{ij}}{\boldsymbol{s}_k^T\widetilde{\boldsymbol{K}}\boldsymbol{s}_k}\left(G_{i_F}(\hat{\boldsymbol{\alpha}}) - G_{j_F}(\hat{\boldsymbol{\alpha}})\right)^2 = 0, \forall i,j. \end{aligned} \tag{38}$$

According to (38), we can obtain the $G_i(\hat{\boldsymbol{\alpha}}) = G_j(\hat{\boldsymbol{\alpha}}), \forall i \neq j$. This implies that the KKT condition holds and the limit point $\hat{\boldsymbol{\alpha}}$ is a global optimal solution of $\mathcal{D}(\boldsymbol{\alpha})$. Since the dual function $\mathcal{D}(\boldsymbol{\alpha})$ is strictly convex, there is a unique global optimal solution. It follows that $\hat{\boldsymbol{\alpha}} = \boldsymbol{\alpha}^*$. □

6 Numerical experiment

The main content of this section is to test the performance of the proposed CFGSMO algorithm. Algorithms used for comparison include first-order SMO (abbreviated as SMO1), second-order SMO (abbreviated as SMO2), FGSMO, CG, and ICG. The kernel function is RBF (Radial basis function), namely $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(-\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2^2/2\sigma^2\right)$, where σ is the kernel width. To compare the efficiency of the algorithms as much as possible, the kernel width σ is not set to an optimal value, but to a set of grid values. Specifically, the value of the kernel width σ is $2^i(-5, \cdots -1, 0, 1, \cdots 5)$, a total of 11 different kernel widths. The stopping condition ϵ is uniformly set to 0.001 [18], and the penalty coefficient γ is $2^i(i = -1, 0, 1, ...12)$, respectively. All algorithms are implemented using MATLAB R2020a and executed on a personal computer

Table 2 Basic information about the datasets

Regression			Regression			Classification			Classification		
Dataset	Size	Dim	Dataset	Size	Dim	Dataset	Size	Dim	Dataset	Size	Dim
mg	1385	7	housing	506	14	a1a	1605	123	w2a	3470	300
mpg	392	8	puma8NH	8192	9	a2a	2265	123	w3a	4912	300
pol	15,000	50	ailerons	7154	42	a3a	3185	123	w4a	7366	300
pyrim	74	28	bodyfat	252	15	a4a	4781	123	w7a	24,692	300
bank32	8192	34	cpusmall	8192	13	a5a	6414	123	heart	270	13
kin8nm	8192	9	space_ga	3107	7	a6a	11220	123	australian	690	14
cadata	17,887	9	triazines	186	61	a7a	16100	123	svmguide1	3089	4
cpu_act	8192	23	eunite2001	336	17	a8a	22696	123	svmguide3	1243	21
abalone	4177	9	cal_housing	20640	10	a9a	32561	123	fourclass	862	2
puma32H	8192	33	delta_ailerons	7129	6	w1a	2477	300	breast_cancer	683	10

with 64 G memory, Inter(R) Xeon(R) W-2123 3.6GHz processor and operating system Ubuntu 20.04.

6.1 Benchmarking datasets

Twenty regression benchmark datasets and twenty classification benchmark datasets in Table 2 are used to test the efficiency of these six algorithms. Among them, except the regression datasets *ailerons*, *cal_housing, bank32, kin8nm, cpu_act, puma32H, puma8NH, delta_ailerons* and *pol* are from LIACC,[1] the rest of the datasets are taken from the LIBSVM Data.[2]

For the regression datasets in Table 2, all are normalized to interval $[-1, 1]$. The benchmark datasets *a4a*, *a5a*, *a6a* and *a7a*, as well as *w2a*, *w3a*, *w4a* and *w7a* are also used to observe the variation in the number of iterations with the benchmark datasets size.

6.2 Execution time comparison

Next, we will compare the execution time of the SMO1, SMO2, FGSMO, CG, ICG and CFGSMO on regression and classification datasets. Tables 3, 4, 5 and 6 record the total execution time of the six algorithms under 11 different kernel widths σ. Table 3 is the execution time comparison of regression datasets *ailerons*, *cal_housing*, *cart_delve* and *pol*. Tables 4, 5 and 6 show the execution time comparison for the binary datasets *a4a*, *a5a*, *a6a*, *a7a*, *a8a*, *a9a*, *w1a*, *w2a*, *w3a*, *w4a*, *w7a* and *svmguide1*.

For the regression datasets *ailerons*, *cal_housing*, *cart_delve* and *pol*, the total execution time of CG and ICG methods is significantly more than that of SMO-type algorithms. In particular, the CG method is significantly less efficient than other algorithms. Although the efficiency of the CG-type method is higher than that of SMO1 when the penalty coefficient γ is large, it is still lower than other SMO-type algorithms. From the total execution time in Table 3, it can be seen that the larger the scale of the datasets, the lower the efficiency of the CG-type method and the higher the efficiency of the SMO-type algorithms. Hence, using the SMO-type algorithms to train the LS-SVM is more efficient than the CG-type method when dealing with large-scale datasets. The SMO-type algorithms are more suitable for the LS-SVM learning task of large-scale datasets. In addition, there are also obvious differences between SMO-type algorithms. Note that, when the penalty coefficient γ is small, the total execution time of SMO1 is significantly less than other SMO-type algorithms. However, when the penalty coefficient γ is gradually increased, the total execution time of SMO1 is gradually more than other SMO-type algorithms. This shows that the SMO1 is not efficient in the face of a large penalty coefficient γ. At the same time, the efficiency of the SMO2 is gradually improved. However, when the penalty coefficient γ increases to a certain extent, the execution time of SMO2 is gradually longer than that of the CFGSMO. This shows that the CFGSMO is significantly more efficient than other SMO algorithms when faced with a larger penalty coefficient γ. Moreover, in other cases, the total execution time of CFGSMO is not significantly different from other SMO-type algorithms.

For the 12 binary classification datasets *a4a*, *a5a*, *a6a*, *a7a*, *a8a*, *a9a*, *w1a*, *w2a*, *w3a*, *w4a*, *w7a* and *svmguide1* in Table 2, their kernel matrices are usually sparse. We know that SMO-type algorithms are very efficient when dealing with sparse matrices. Hence, when dealing with classification tasks, the learning efficiency of the LS-SVM model

[1] https://www.dcc.fc.up.pt/∼ltorgo/Regression/DataSets.html.

[2] https://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/.

Table 3 The total execution time of the SMO1, SMO2, FGSMO, CG, ICG and CFGSMO algorithms on the regression benchmark datasets *ailerons*, *cal_housing*, *cadata* and *pol* (Units: seconds). Boldface is used to highlight the shortest time, similarly hereinafter

ailerons							cal_housing					
$log2^{\gamma}$	SMO1	SMO2	FGSMO	CG	ICG	CFGSMO	SMO1	SMO2	FGSMO	CG	ICG	CFGSMO
− 1	**5.26**	11.45	12.64	35.84	37.51	17.27	**13.54**	27.69	30.69	100.22	94.12	42.34
0	**6.27**	12.41	13.56	48.09	44.17	18.22	**16.96**	28.62	31.50	130.76	112.17	43.14
1	**8.22**	14.20	15.65	63.10	52.37	19.99	**23.73**	30.89	33.83	173.60	137.16	45.26
2	**11.81**	17.74	19.28	84.87	64.93	23.27	36.56	**34.79**	38.26	230.39	173.50	49.42
3	**18.17**	22.84	25.06	112.54	81.29	28.77	61.32	**41.85**	46.39	311.15	224.46	56.29
4	**30.68**	31.46	34.67	152.12	103.34	37.98	110.98	**55.11**	60.26	419.19	291.71	69.77
5	53.58	**45.72**	50.22	208.49	137.98	52.22	207.55	**79.42**	87.67	579.65	391.74	94.04
6	96.04	**69.39**	76.66	284.61	186.14	75.97	396.71	**124.83**	136.93	796.75	536.63	137.14
7	173.43	**109.72**	121.12	401.90	257.51	114.61	779.53	**211.50**	231.94	1091.08	739.38	217.93
8	315.71	178.78	197.39	562.20	362.14	**178.63**	1525.40	371.43	407.74	1543.86	1034.72	**362.78**
9	580.29	297.74	328.53	785.93	515.90	**285.66**	2976.40	668.82	731.13	2174.78	1465.88	**632.45**
10	1061.32	501.39	552.14	1107.23	727.08	**466.19**	5821.66	1221.41	1345.80	3050.19	2076.08	**1127.69**
11	1938.31	851.41	943.71	1591.40	1033.15	**773.63**	11422.21	2251.12	2469.47	4289.49	2982.97	**2049.17**
12	3551.46	1466.08	1619.64	2233.46	1470.27	**1294.09**	2,2350.54	4149.63	4557.02	6149.46	4257.50	**3753.15**
cadata							pol					
− 1	**10.85**	22.43	24.59	66.97	67.74	34.41	**7.88**	16.98	18.75	53.40	51.51	26.15
0	**13.40**	22.73	24.94	86.87	78.88	34.82	**9.16**	17.75	19.37	70.21	61.52	26.68
1	**17.81**	24.01	26.48	113.15	97.70	36.12	**11.81**	18.97	20.90	92.04	75.09	27.87
2	26.36	**26.33**	28.87	150.60	119.53	38.18	**17.16**	21.59	23.60	123.09	93.02	30.41
3	43.63	**30.15**	32.98	199.23	149.90	42.38	27.22	**26.55**	29.04	167.30	119.55	34.90
4	78.03	**37.87**	41.58	268.99	197.51	50.26	45.63	**35.70**	39.22	229.93	158.44	42.86
5	146.52	**52.03**	56.35	363.72	264.36	63.79	81.44	**51.61**	56.68	315.91	212.96	57.18
6	284.63	**76.72**	83.86	502.04	350.87	87.82	149.56	**80.13**	87.85	436.41	295.01	81.93
7	556.95	**125.24**	138.89	683.06	485.04	133.32	281.70	129.44	140.58	612.26	414.12	**124.37**
8	1098.88	219.77	239.74	953.58	684.58	**216.67**	526.39	213.70	233.17	863.17	583.13	**198.54**
9	2158.33	396.83	438.23	1320.89	971.32	**378.14**	984.82	362.37	394.76	1222.95	823.54	**328.43**
10	4279.72	729.77	804.46	1878.30	1340.62	**675.33**	1853.03	624.76	683.70	1691.76	1171.89	**553.22**
11	8406.90	1370.17	1506.00	2644.40	1967.38	**1236.31**	3471.34	1084.68	1186.06	2439.46	1680.71	**955.79**
12	16550.26	2599.86	2860.40	3721.15	2773.77	**2345.83**	6467.86	1895.67	2076.84	3475.46	2451.36	**1666.51**

using the SMO-type algorithms is usually higher than that of the CG-type algorithms. The total execution times in Tables 4– 6 confirm this conclusion. As with the regression datasets, when the penalty coefficient γ is small, the total execution time of the SMO1 is slightly less than that of the SMO2, FGSMO and CFGSMO and significantly less than that of the CG and ICG algorithms. Therefore, when γ is small, it is efficient to use SMO1 to solve the LS-SVM. But when γ is large, the first-order SMO is no longer suitable for training the LS-SVM. At this time, the efficiency of the SMO algorithm using the second-order information is significantly higher than that of the first-order SMO. Although the SMO2, FGSMO and CFGSMO algorithms all use second-order information, from the numerical results in Tables 4–6, the CFGSMO algorithm is obviously more suitable for the case where the penalty coefficient γ is large. Hence, when the penalty coefficient γ is large, using the CFGSMO algorithm to train the LS-SVM will be faster and more efficient. Similarly, when dealing with large-scale binary datasets, SMO-type algorithms also have advantages over CG-type algorithms. This shows that SMO-type algorithms are usually more efficient than CG-type algorithms for both classification and regression tasks. In addition, in the SMO-type algorithms, the CFGSMO algorithm will have more competitive advantages.

6.3 Comparison of iterative processes

Lemma 5.1 points out that the functional gain of each iteration of the CFGSMO is greater than that of the

Table 4 The total execution time of the SMO1, SMO2, FGSMO, CG, ICG and CFGSMO algorithms on the classification benchmark datasets *a4a*, *a5a*,*a6a*, and *a7a* (Units: seconds)

a4a							a5a					
$log2^{\gamma}$	SMO1	SMO2	FGSMO	CG	ICG	CFGSMO	SMO1	SMO2	FGSMO	CG	ICG	CFGSMO
− 1	**1.35**	3.04	3.27	2.27	3.83	3.96	**2.23**	5.02	5.35	4.61	6.76	6.74
0	**1.55**	3.10	3.33	2.97	4.10	4.07	**2.47**	5.09	5.42	5.88	7.55	6.69
1	**1.83**	3.47	3.62	4.05	4.73	4.32	**2.95**	5.64	5.99	7.41	8.72	7.26
2	**2.32**	3.89	4.11	5.20	5.49	4.68	**3.91**	6.58	6.82	9.75	10.03	7.97
3	**3.28**	4.61	4.83	6.51	6.50	5.35	**5.27**	7.55	7.99	12.24	11.96	9.14
4	**4.76**	5.70	6.04	8.71	8.13	6.26	**7.87**	9.30	10.00	15.97	14.80	10.91
5	**7.27**	7.34	7.82	10.91	9.71	7.74	12.32	**12.00**	12.91	21.13	18.36	13.22
6	11.19	**9.74**	10.33	14.31	11.99	9.86	19.80	**16.06**	17.30	27.58	23.74	16.97
7	18.08	13.36	14.03	17.95	15.11	**12.97**	32.20	22.26	23.98	36.20	29.92	**22.13**
8	29.83	18.58	19.70	22.87	19.41	**17.25**	53.46	31.49	33.68	46.51	38.19	**30.12**
9	49.32	26.12	27.93	30.54	24.84	**23.32**	91.59	45.08	48.05	59.46	49.12	**40.64**
10	84.81	36.66	38.44	39.37	31.79	**30.92**	158.57	64.60	69.02	77.68	64.03	**55.73**
11	148.33	50.99	54.34	49.64	41.55	**41.48**	280.72	92.03	98.26	100.10	84.53	**75.88**
12	264.05	70.46	74.60	65.61	56.22	**55.37**	509.87	129.81	138.95	127.42	108.70	**103.26**
a6a							a7a					
− 1	**5.95**	13.95	15.41	17.18	22.35	21.21	**10.79**	24.86	26.82	40.37	48.10	37.55
0	**6.49**	14.20	15.69	22.52	25.70	21.58	**12.41**	26.55	28.50	53.40	56.29	39.50
1	**8.03**	15.99	17.59	29.22	29.82	23.11	**15.50**	29.60	31.78	69.99	66.98	42.21
2	**10.58**	17.64	19.65	38.25	35.47	25.16	**20.58**	31.93	35.06	91.26	81.52	45.26
3	**14.84**	20.65	22.96	49.34	43.79	28.34	**30.02**	38.89	42.48	118.24	100.59	52.59
4	**22.58**	25.53	28.42	64.57	55.23	33.58	**46.11**	48.63	53.22	155.59	128.67	63.65
5	36.17	**33.15**	36.69	85.15	69.88	41.51	74.51	**63.77**	69.87	203.81	164.61	79.09
6	60.22	**45.11**	50.18	113.74	89.69	53.45	124.80	**87.64**	95.90	266.64	216.13	102.31
7	99.81	**63.29**	70.43	147.27	117.79	71.12	212.29	**123.92**	136.35	358.97	288.55	138.18
8	169.27	**91.96**	102.07	193.30	156.50	97.78	365.05	**181.89**	198.88	470.46	376.72	191.11
9	293.33	**134.98**	149.96	251.99	204.49	136.29	639.01	270.10	297.45	635.89	516.79	**268.39**
10	522.70	199.64	221.72	338.35	276.13	**191.56**	1150.24	406.70	444.41	831.40	683.26	**383.34**
11	946.96	294.62	329.04	437.25	367.85	**269.41**	2121.40	609.36	666.82	1087.84	885.29	**547.41**
12	1733.70	431.43	480.78	574.46	477.75	**376.20**	3903.18	912.49	1008.44	1425.06	1184.29	**787.72**

Boldface is used to highlight the shortest time

FGSMO. Naturally, it is also larger than other SMO-type algorithms. Because the functional gain of the SMO2 is greater than that of the SMO1. And SMO2 is a special case of FGSMO, which naturally satisfies the case of Lemma 5.1. Figures 2 and 3 show the iterative process of regression dataset *pol* and classification dataset *w1a* under different kernel width σ, respectively. Notice that, the results in Figs. 2 and 3 are only the results of the first 2000 iterations. The "Dual" in Figs. 2 and 3 denotes the function value of the dual objective function (7). It can be clearly seen from Figs. 2 and 3 that the convergence speed of the CFGSMO algorithm is significantly faster than that of the other three SMO algorithms. Second only to the CFGSMO are SMO2 and FGSMO, and the slowest convergence speed is SMO1. No matter in the regression dataset *pol* or the classification dataset *w1a*, the functional gain of the CFGSMO algorithm is significantly more than that of other SMO-type algorithms. However, with the continuous increase in the kernel width σ, the gap between the four SMO algorithms is gradually narrowed, because in the process of gradually reducing the σ, the SMO2 will gradually degenerate into the form of the SMO1.

Figure 4 compares the average number of iterations for the four SMO-type algorithms with different training set sizes and different kernel width σ. Hence, according to the basic information of the datasets in Table 2, we selected two groups datasets. The first group is *a4a*, *a5a*, *a6a* and *a7a*, and the second group is *w1a*, *w2a*, *w3a*, *w4a* and *w7a*. Note that, the sample sizes of both datasets do not grow strictly linearly.

Table 5 The total execution time of the SMO1, SMO2, FGSMO, CG, ICG and CFGSMO algorithms on the classification benchmark datasets *a8a*, *a9a*, *svmguide1*, and *w1a* (Units: seconds)

a8a							a9a					
$log2^{\gamma}$	SMO1	SMO2	FGSMO	CG	ICG	CFGSMO	SMO1	SMO2	FGSMO	CG	ICG	CFGSMO
− 1	**20.22**	46.76	50.65	88.30	100.81	69.49	**38.01**	88.80	95.03	211.55	218.88	130.23
0	**22.91**	49.55	53.20	118.40	118.86	71.93	**43.70**	95.86	100.71	280.98	265.10	136.22
1	**28.80**	54.44	59.11	154.99	142.43	78.30	**55.82**	106.28	110.71	373.10	318.59	146.31
2	**38.91**	61.40	67.04	203.13	178.41	86.11	**76.39**	119.75	126.58	485.14	395.66	165.18
3	**57.28**	71.83	78.89	264.64	222.84	97.89	**113.75**	142.35	151.78	641.72	503.40	188.21
4	89.12	**88.10**	96.77	352.53	277.50	114.82	180.77	**177.47**	189.59	858.86	653.03	227.39
5	146.37	**115.45**	126.21	462.84	366.80	144.07	301.92	**233.98**	248.54	1137.99	857.76	281.77
6	246.92	**159.84**	175.22	614.29	485.42	189.28	517.83	**325.96**	344.53	1512.93	1141.98	369.77
7	426.78	**230.66**	253.43	820.70	641.57	259.48	896.59	**474.15**	501.49	2013.71	1548.24	508.64
8	734.96	**341.06**	374.07	1073.52	857.24	358.55	1600.47	722.14	755.49	2733.10	2118.99	**721.00**
9	1309.10	**513.50**	564.49	1425.43	1135.50	515.29	2883.69	1105.86	1153.90	3706.57	2839.44	**1040.62**
10	2391.88	781.33	854.28	1914.71	1569.48	**738.75**	5329.66	1705.49	1775.86	4898.03	3764.89	**1528.19**
11	4412.57	1189.24	1308.61	2510.24	2068.94	**1083.15**	9955.06	2581.12	2719.47	6371.45	5119.19	**2210.63**
12	8249.48	1808.33	1985.43	3265.98	2699.80	**1576.62**	18716.80	3994.65	4194.80	8575.02	6931.60	**3273.39**
svmguide1							w1a					
− 1	**0.93**	1.82	1.84	0.53	1.43	2.20	**0.40**	0.77	0.79	0.58	0.99	0.93
0	**0.90**	1.79	1.87	0.68	1.52	2.27	**0.43**	0.75	0.76	0.75	1.08	0.88
1	**0.92**	1.91	1.92	0.86	1.61	2.36	**0.46**	0.76	0.77	0.88	1.18	0.91
2	**0.96**	1.87	1.97	1.12	1.66	2.33	**0.51**	0.78	0.80	1.13	1.33	0.93
3	**1.19**	1.94	2.04	1.29	1.86	2.46	**0.69**	0.87	0.88	1.43	1.49	0.98
4	**1.26**	2.05	2.14	1.67	2.22	2.56	**0.85**	0.91	0.97	1.68	1.69	1.04
5	**1.62**	2.18	2.28	2.03	2.29	2.69	**0.85**	1.03	1.06	2.05	1.92	1.14
6	**1.85**	2.32	2.46	2.95	2.79	2.70	**1.16**	1.16	1.18	2.57	2.32	1.24
7	**2.22**	2.56	2.68	3.34	3.42	2.88	2.74	**1.32**	1.37	3.16	2.69	1.37
8	**2.67**	2.86	3.00	4.30	3.97	3.13	3.98	1.57	1.59	3.82	3.26	**1.57**
9	3.49	**3.27**	3.46	5.89	4.54	3.41	3.23	1.80	1.87	5.08	4.16	**1.75**
10	4.60	3.79	4.02	7.13	5.70	**3.80**	8.95	2.10	2.16	6.07	5.00	**1.97**
11	6.12	4.47	4.71	9.02	6.88	**4.17**	5.63	2.46	2.52	7.78	6.18	**2.16**
12	8.01	5.24	5.55	11.30	8.45	**4.77**	23.59	2.88	2.93	9.14	7.48	**2.50**

Boldface is used to highlight the shortest time

From the results shown in Fig. 4, we can know that with the gradual increase in the training datasets size, the average number of iterations of the four SMO-type algorithms increases approximately linearly. Among them, the growth rate of SMO1 is the fastest, and the growth rate of CFGSMO algorithm is the slowest. This is because, in each iteration, the functional gain of the CFGSMO algorithm is greater than that of the other three SMO-type algorithms.

6.4 Comparison of accuracy

In this section, we will test the accuracy of CFGSMO. Similarly, the choice of kernel function is still RBF. The range of γ is $2^{-5}, 2^{-3}, \cdots, 2^{9}, 2^{11}$, and the range of kernel width σ is $2^{-11}, 2^{-9}, \cdots, 2^{1}, 2^{3}$. The evaluation criterion for classification is

$$\text{Acc} = \frac{\text{Number of correctly predicted labels}}{\text{total testing labels}} \times 100\%. \tag{39}$$

The mean squared error (MSE) is used to evaluate the performance of the regression, that is

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{40}$$

where $\hat{y}_i$ denotes the predicted target value. Tables 7 and 8 are the result of fivefold cross-validation.

Table 6 The total execution time of the SMO1, SMO2, FGSMO, CG, ICG and CFGSMO algorithms on the classification benchmark datasets *w2a*, *w3a*, *w4a*, and *w7a* (Units: seconds)

w2a							w3a					
$log2^{\gamma}$	SMO1	SMO2	FGSMO	CG	ICG	CFGSMO	SMO1	SMO2	FGSMO	CG	ICG	CFGSMO
− 1	**0.80**	1.57	1.65	1.30	1.92	2.08	**1.29**	2.97	3.14	2.92	3.94	3.84
0	**0.86**	1.60	1.69	1.69	2.18	2.00	**1.48**	3.09	3.20	3.47	4.26	3.89
1	**0.95**	1.69	1.72	2.07	2.34	2.04	**1.64**	3.11	3.28	4.32	4.98	3.97
2	**1.02**	1.81	1.80	2.66	2.70	2.13	**1.85**	3.25	3.42	5.58	5.64	4.16
3	**1.36**	1.86	1.95	3.22	3.13	2.23	**2.36**	3.53	3.75	6.85	6.88	4.41
4	**1.65**	2.18	2.20	4.04	3.71	2.50	**2.70**	3.91	4.14	8.68	8.17	4.81
5	**1.79**	2.32	2.40	5.15	4.42	2.63	**3.30**	4.36	4.64	10.86	9.63	5.28
6	3.09	**2.60**	2.76	6.35	5.41	2.92	**5.00**	5.08	5.36	14.38	11.76	5.92
7	6.34	**3.02**	3.14	7.83	6.49	3.32	10.77	**6.02**	6.18	17.41	14.58	6.64
8	7.49	**3.60**	3.73	9.63	8.23	3.74	15.53	**7.11**	7.51	22.78	17.55	7.67
9	12.74	**4.23**	4.45	13.03	10.51	4.27	16.87	**8.66**	9.06	28.73	23.35	8.77
10	20.46	5.09	5.32	15.26	13.03	**4.87**	36.84	10.42	10.94	36.92	29.81	**10.14**
11	18.22	6.00	6.31	20.21	16.05	**5.54**	36.65	12.51	13.10	46.39	37.52	**11.75**
12	16.42	7.07	7.42	24.17	19.56	**6.38**	40.48	15.03	15.74	56.64	48.12	**13.69**
w4a							w7a					
− 1	**2.37**	5.65	5.82	6.87	9.14	7.44	**20.32**	48.08	52.07	118.33	127.15	72.10
0	**2.55**	5.67	6.00	8.51	10.16	7.70	**21.86**	48.05	51.44	153.37	147.72	71.13
1	**2.82**	5.89	6.24	10.90	11.96	7.92	**24.80**	50.39	54.35	201.08	178.55	74.13
2	**3.42**	5.94	6.36	13.70	13.65	8.00	**28.75**	54.38	59.29	263.19	218.80	79.97
3	**4.17**	6.39	6.82	17.54	16.48	8.49	**36.44**	59.47	64.52	340.39	272.35	85.97
4	**5.36**	7.07	7.49	22.28	20.04	9.09	**43.59**	67.20	72.69	444.21	339.26	94.52
5	**5.81**	8.09	8.54	29.25	25.38	10.17	**56.10**	77.75	84.84	574.94	439.50	105.82
6	**9.13**	9.59	10.08	36.56	29.88	11.35	100.52	**93.64**	101.51	758.17	574.96	121.70
7	14.31	**11.52**	12.13	47.16	38.63	13.27	147.37	**116.73**	126.99	978.69	733.77	144.97
8	19.79	**14.16**	14.82	61.10	47.60	15.14	181.07	**148.33**	162.89	1293.99	985.21	175.18
9	22.09	**17.57**	18.10	77.30	60.28	17.86	245.04	**190.92**	209.30	1694.07	1310.89	210.94
10	49.05	21.83	22.77	100.38	78.82	**21.25**	387.07	**252.60**	274.72	2250.34	1746.16	260.26
11	46.71	27.06	28.53	127.03	99.51	**25.42**	516.75	331.51	357.90	2968.53	2363.92	**326.27**
12	93.26	33.37	34.89	161.41	128.57	**29.83**	798.13	432.46	464.82	3801.45	3080.61	**409.29**

Boldface is used to highlight the shortest time

The results in Tables 7 and 8 show that the cross-validation accuracy of SMO1, SMO2, FGSMO and CFGSMO is not much different. Although the accuracy of CFGSMO is sometimes lower than one of SMO1, SMO2 and FGSMO, it is not the worst one. In Table 8, CFGSMO has the smallest MSE on the *bodyfat*, *delta_ailerons*, *mpg*, *puma8NH*, *pyrim* and *triazines* datasets, and the accuracy is not bad on other datasets. In terms of the hyper-parameters, the results of γ^* and σ^* selection for four SMO algorithms are mostly the same, especially for the regression datasets. Overall, the accuracy of CFGSMO is not much worse than SMO1, SMO2 and FGSMO.

6.5 Short summary and discussion

At the $k-$th iteration, SMO1 and SMO2 need $2n$ flops (floating point operations) to update the working set, and the update of gradient $\boldsymbol{G}(\boldsymbol{\alpha})$ requires n flops, which requires about $3n$ iteration costs. Different from the SMO1 and SMO2, FGSMO needs to compute the absolute value of gradient $\boldsymbol{G}(\boldsymbol{\alpha})$ when updating the working set. If $n_{\boldsymbol{\alpha}}(0 \leq n_{\boldsymbol{\alpha}} \leq n)$ is used to represent the number of elements less than 0 in the gradient $\boldsymbol{G}(\boldsymbol{\alpha})$, the FGSMO update working set requires $2n + n_{\boldsymbol{\alpha}}$ flops, and the update of the gradient $\boldsymbol{G}(\boldsymbol{\alpha})$ requires n flops. In total, about $3n + n_{\boldsymbol{\alpha}}$ flops are required. For CFGSMO, updating the working set requires $2n + n_{\boldsymbol{\alpha}}$ flops, and updating the gradient $\boldsymbol{G}(\boldsymbol{\alpha})$

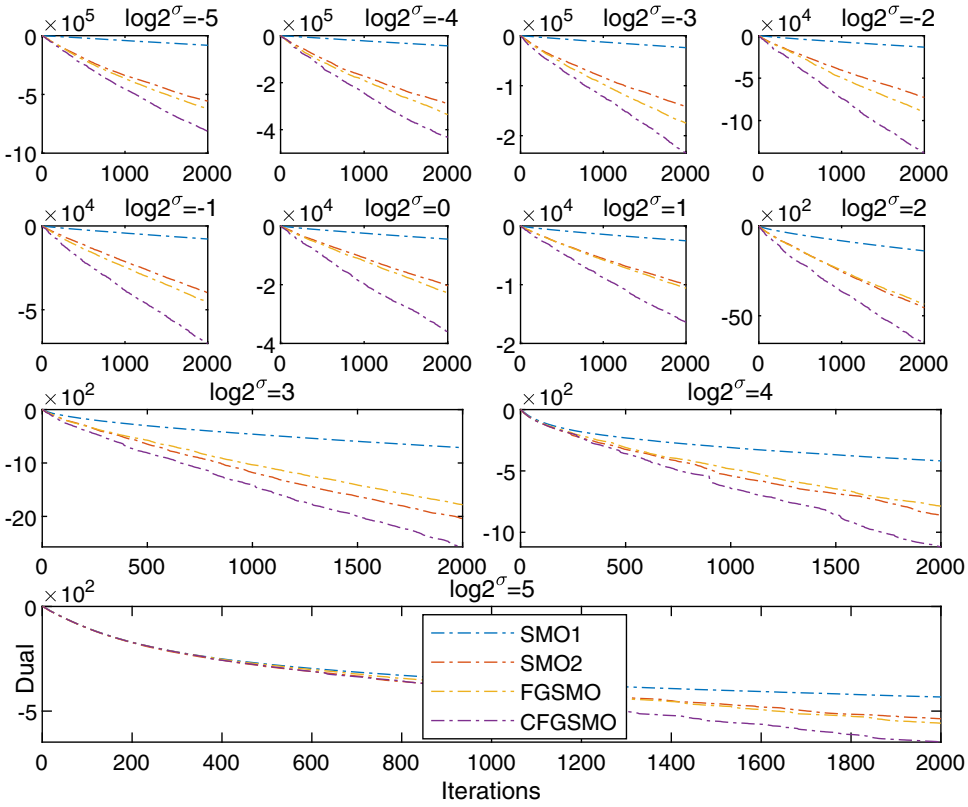

Fig. 2 The iterative process of regression dataset *pol* under different kernel width σ

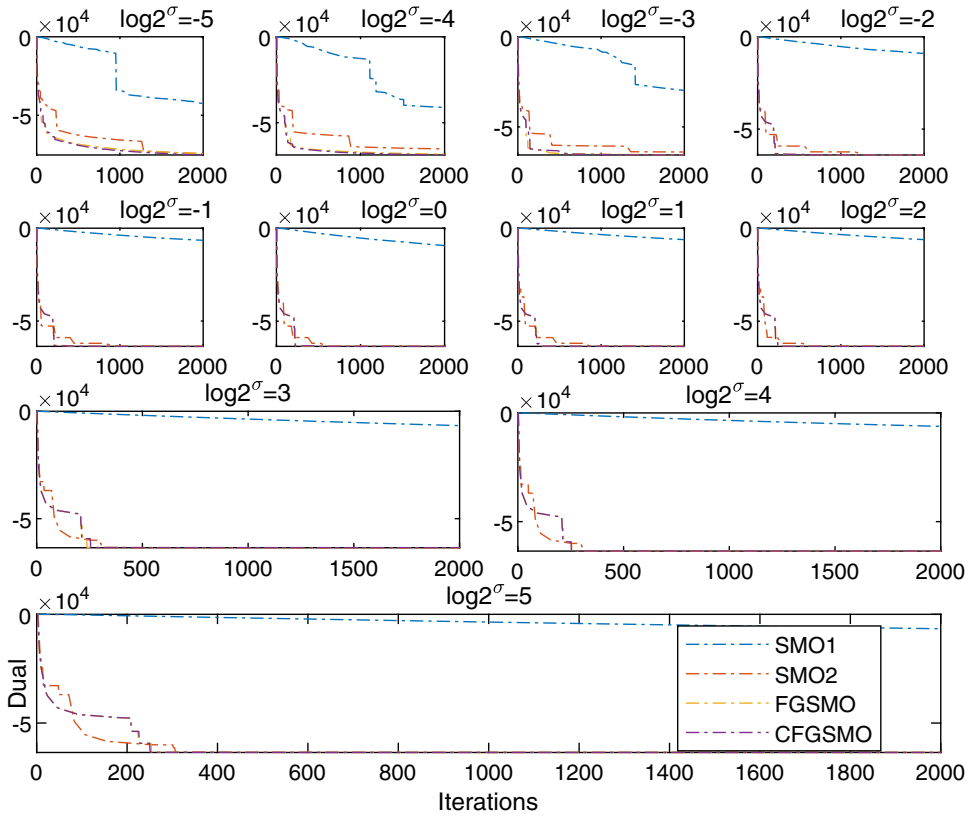

Fig. 3 The iterative process of classification dataset *w1a* under different kernel width σ

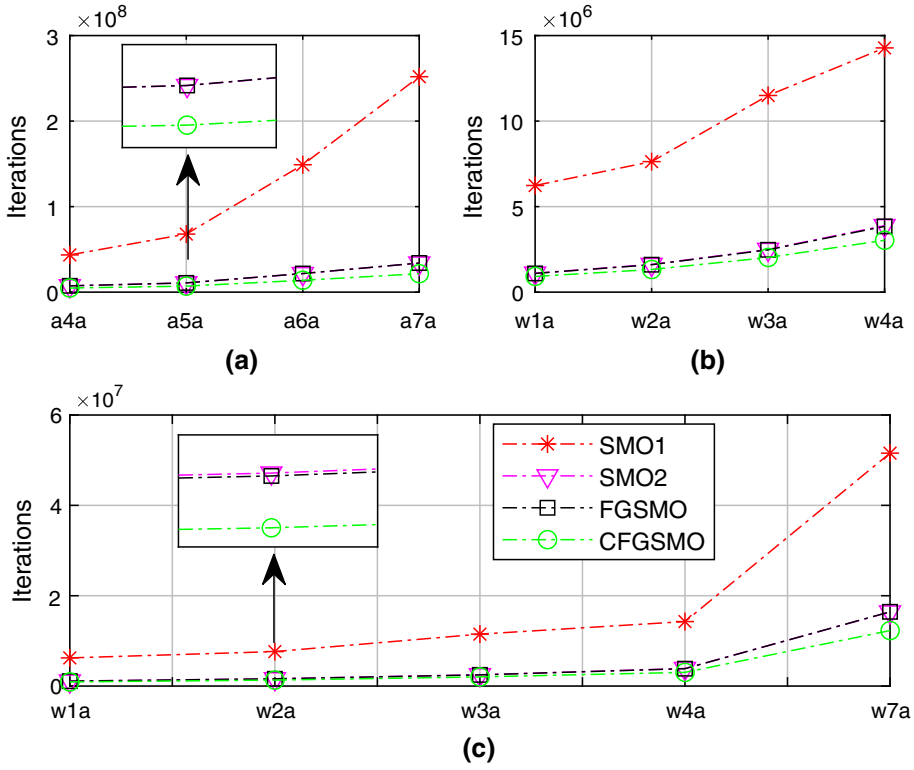

Fig. 4 Variation in the number of iterations with training set size

Table 7 Cross-validation results for hyper-parameters. γ^* and σ^* (log2-scale) denote the optimal hyper-parameters, and Acc* denotes the best accuracy for cross-validation (classification)

Datasets	SMO1			SMO2			FGSMO			CFGSMO		
	γ^*	σ^*	Acc* (%)	γ^*	σ^*	Acc* (%)	γ^*	σ^*	Acc* (%)	γ^*	σ^*	Acc* (%)
a1a	5	3	82.368	7	3	82.928	7	3	82.741	7	3	82.492
a2a	5	1	81.413	5	1	82.252	5	1	81.634	5	1	82.119
a3a	7	3	82.983	7	3	83.579	5	1	83.234	5	1	83.862
a4a	5	1	84.124	5	1	83.686	5	1	83.874	5	1	82.744
a5a	5	1	83.942	5	1	84.004	5	1	84.050	5	1	83.645
australian	1	− 1	86.667	1	− 1	86.377	1	3	85.797	1	− 1	87.246
breast_cancer	11	− 5	97.656	9	− 7	97.369	9	− 7	97.363	11	− 5	97.514
fourclass	− 5	3	98.726	− 5	3	98.956	− 5	3	98.839	− 5	3	98.957
heart	1	1	84.074	3	− 1	85.185	5	1	84.444	7	1	84.074
w1a	5	− 1	97.618	5	− 1	97.577	5	− 1	97.336	5	− 1	97.496
w2a	7	− 1	97.349	7	− 1	97.262	7	− 1	97.205	7	− 1	97.205
w3a	5	− 1	97.252	5	− 1	97.252	5	− 1	97.272	5	− 1	97.252
w4a	5	− 1	97.122	5	− 1	97.095	5	− 1	97.108	5	− 1	97.095
svmguide1	− 3	1	94.982	− 3	1	94.723	− 3	1	94.756	− 3	1	94.755
svmguide3	1	3	83.265	1	3	83.669	1	3	83.912	1	3	83.269

requires n flops. If n_s and n_t are used to represent the number of nonzero elements in the vectors $\boldsymbol{s}$ and $\boldsymbol{t}$, respectively, then the update of the vectors $\boldsymbol{s}$ and $\boldsymbol{\alpha}$ requires $2n_s$ flops, and the update of the vector $\boldsymbol{t}$ requires n_t flops. Since the number of nonzero elements in the vector $\boldsymbol{s}$ is much less than n, $n_t \approx n$. Hence, the cost of each iteration of CFGSMO is about $3n + n_{\boldsymbol{\alpha}} + 2n_s + n_t \approx 4n + n_{\boldsymbol{\alpha}}$. Note that, $4n \leq 4n + n_{\boldsymbol{\alpha}} \leq 5n$. Compared with SMO1, SMO2 and

Table 8 Cross-validation results for hyper-parameters. γ^* and σ^* (log2-scale) denote the optimal hyper-parameters, and MSE* denotes the minimum MSE for cross-validation (regression)

Datasets	SMO1			SMO2			FGSMO			CFGSMO		
	γ^*	σ^*	MSE*	γ^*	σ^*	MSE*	γ^*	σ^*	MSE*	γ^*	σ^*	MSE*
abalone	− 5	3	0.263	− 5	3	0.266	− 5	3	0.263	− 5	3	0.266
ailerons	9	3	0.138	9	3	0.137	9	3	0.132	9	3	0.135
bank32	5	1	0.043	5	1	0.056	5	1	0.044	5	1	0.046
bodyfat	1	1	0.013	1	3	0.009	1	3	0.010	1	3	0.008
cpu_act	1	− 3	0.010	1	− 3	0.010	1	− 3	0.010	1	− 3	0.010
cpusmall	1	− 1	0.008	1	− 1	0.007	1	− 1	0.010	1	− 1	0.009
delta_ailerons	1	− 1	0.007	− 1	− 3	0.007	1	− 1	0.007	1	− 1	0.006
eunite2001	1	− 1	0.034	1	− 1	0.034	1	− 1	0.033	1	− 1	0.033
housing	1	3	0.024	1	3	0.025	1	3	0.025	1	3	0.028
kin8nm	1	1	0.023	1	1	0.023	1	1	0.022	1	1	0.022
mg	− 3	− 1	0.125	− 3	− 1	0.127	− 3	− 1	0.125	− 3	− 1	0.126
mpg	3	− 1	0.038	3	− 1	0.038	3	− 1	0.039	3	− 1	0.038
puma32H	3	3	0.077	1	3	0.078	3	3	0.082	3	3	0.080
puma8NH	1	1	0.081	1	1	0.084	1	1	0.083	1	1	0.081
pyrim	3	3	0.054	3	3	0.065	3	3	0.057	3	3	0.049
space_ga	− 1	3	1.020	− 1	3	1.025	− 1	3	1.027	− 1	3	1.033
triazines	5	3	0.245	5	3	0.224	5	3	0.235	5	3	0.213

FGSMO, the cost of each iteration of CFGSMO increases about 25% to 40% of the computation. This shows that the advantages of CFGSMO are only evident when the total number of iterations of CFGSMO is less than 60% to 75% of the other three SMO algorithms.

From the numerical results in Table 3 to Table 6, it can be seen that the SMO-type algorithm with second-order information is more suitable for training the LS-SVM model than the CG-type method. As the penalty parameter increases gradually, the efficiency of SMO2, FGSMO and CFGSMO gradually increases, while the efficiency of SMO1 and CG-type algorithms gradually decreases. Note that, equation (14) can be further rewritten as

$$f_G\left(\Delta\boldsymbol{\alpha}^k\right) = \frac{\left(G_j\left(\boldsymbol{\alpha}^k\right) - G_i\left(\boldsymbol{\alpha}^k\right)\right)^2}{2\left(\frac{2}{\gamma} + \kappa(i,j)\right)}, \tag{41}$$

where $\kappa(i,j) = [\boldsymbol{K}]_{ii} + [\boldsymbol{K}]_{jj} - [\boldsymbol{K}]_{ij} - [\boldsymbol{K}]_{ji}$. The SMO1 algorithm ignores the influence of $\kappa(i,j)$ on $f_G\left(\Delta\boldsymbol{\alpha}^k\right)$. When γ is small, $\kappa(i,j)$ has less influence on $f_G\left(\Delta\boldsymbol{\alpha}^k\right)$, and $\kappa(i,j)$ can be ignored at this time. When γ is large, if $\kappa(i,j)$ is ignored, $f_G\left(\Delta\boldsymbol{\alpha}^k\right)$ will be severely affected. Hence, the SMO1 algorithm is effective when γ is small and gradually becomes less efficient when γ becomes large. However, the FGSMO and SMO2 algorithms consider the influence of $\kappa(i,j)$ on $f_G\left(\Delta\boldsymbol{\alpha}^k\right)$. When γ gradually becomes smaller, FGSMO and SMO2 are gradually equivalent to SMO1, but FGSMO and SMO2 increase some kernel evaluation, so the computational efficiency is slightly lower than that of SMO1. When γ is large, $f_G\left(\Delta\boldsymbol{\alpha}^k\right)$ is greatly affected by $\kappa(i,j)$. At this time, the functional gain $f_G\left(\Delta\boldsymbol{\alpha}^k\right)$ of FGSMO and SMO2 far exceeds that of SMO1.

For the CFGSMO, when γ is small, the efficiency of this algorithm will be slightly lower than other SMO algorithms, but for larger γ, the efficiency of CFGSMO algorithm is the highest. This may be because CFGSMO uses the descent direction of FGSMO to construct a feasible conjugate descent direction in the iterative process and retains the algorithm characteristics of FGSMO under different parameters γ. But CFGSMO increases the functional gain of FGSMO. Therefore, in the face of larger γ, the efficiency of CFGSMO is higher than that of FGSMO and SMO2. This indicates that the execution time of the program will be drastically reduced if CFGSMO is used for a wide range of cross-validation.

7 Conclusion

This work proposes a fast training algorithm for LS-SVM, namely the conjugate functional gain SMO algorithm, and theoretically proves its asymptotic convergence. This algorithm is based on the conjugate direction method and the SMO-type algorithm, which is a combination of these two algorithms. Theoretically, at each iteration, the functional gain of this algorithm is greater than or equal to that of the SMO1, SMO2 and FGSMO algorithms.

From the numerical results, the CFGSMO algorithm is indeed more efficient, especially when γ is large. The larger γ, the more obvious the advantage of CFGSMO. When γ is

small, the efficiency of CFGSMO is slightly lower than that of other SMO algorithms. It may be because the cost of each iteration of CFGSMO is higher than that of other SMO algorithms. In contrast, the efficiency of SMO1 is highest when γ is small, and the smaller γ is the higher the efficiency of SMO1. SMO2 and FGSMO are suitable for moderately sized γ (no more than about 1000). In addition, on large-scale datasets, the efficiency of SMO-type algorithms is significantly higher than that of CG-type, especially CFGSMO. For small- and medium-sized datasets, the efficiency of CG and SMO is not much different. Since CG needs to manipulate the entire kernel matrix every iteration, SMO only needs to manipulate two columns of the kernel matrix. Therefore, SMO-type algorithms may be more efficient than CG-type when training large-scale LS-SVM. In terms of hyper-parameter selection, the optimal hyper-parameters determined by cross-validation of the four SMO algorithms are consistent in most cases, and the performances (Acc and MSE) are not much different.

Since the dual problem of LS-SVM does not contain the box constraint $0 \leq \alpha_i \leq C, \forall i$. Therefore, CFGSMO does not have a clipping and restart step like CSMO. In comparison, the iterative format of CFGSMO will be simpler and easier to implement and will not suffer from the effects of the bounds of inequalities.

Reviewing the computation flow of CFGSMO, it can be seen that the working set selection strategy of CFGSMO is not strictly limited to FGWSS. This suggests that the efficiency of CFGSMO may be further improved if other more efficient working set selection strategies are used instead of FGWSS. In addition, CFGSMO increases the computational cost by about 1/4–2/5 compared with plain SMO-type algorithms. Therefore, how to reduce the computational cost of CFGSMO on the premise of ensuring the efficiency of CFGSMO is also one of the work that needs to be further studied in the future. Besides, extending the conjugate SMO to other support vector machine models is also the next work to be considered.

Acknowledgements This research was supported by the Graduate Research and Innovation Foundation of Chongqing, China (CYS22074), the National Natural Science Foundation of China (71901184), Humanities and Social Science Fund of Ministry of Education of China (19YJCZH119).

Data availability The authors declare that data supporting the findings of this study are available within the article.

Declarations

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

References

1. Burges CJ (1998) A tutorial on support vector machines for pattern recognition. Data Min Knowl Discov 2(2):121–167
2. Smola AJ, Schölkopf B (2004) A tutorial on support vector regression. Stat Comput 14(3):199–222
3. Chen PH, Fan RE, Lin CJ (2006) A study on smo-type decomposition methods for support vector machines. IEEE Trans Neural Netw 17(4):893–908
4. Suykens JA, Vandewalle J (1999) Least squares support vector machine classifiers. Neural Process Lett 9(3):293–300
5. Jiao L, Bo L, Wang L (2007) Fast sparse approximation for least squares support vector machine. IEEE Trans Neural Netw 18(3):685–697
6. Yang X, Lu J, Zhang G (2010) Adaptive pruning algorithm for least squares support vector machine classifier. Soft Comput 14(7):667–680
7. Li B, Song S, Li K (2013) A fast iterative single data approach to training unconstrained least squares support vector machines. Neurocomputing 115:31–38
8. Xia X-L (2018) Training sparse least squares support vector machines by the qr decomposition. Neural Netw 106:175–184
9. Chua KS (2003) Efficient computations for large least square support vector machine classifiers. Pattern Recognit Lett 24(1–3):75–80
10. Suykens J, Lukas L, Van Dooren P, De Moor B, Vandewalle J, *et al.* (1999) Least squares support vector machine classifiers: a large scale algorithm. In: European Conference on Circuit Theory and Design, ECCTD, vol. 99, pp. 839–842. Citeseer
11. Chu W, Ong CJ, Keerthi SS (2005) An improved conjugate gradient scheme to the solution of least squares svm. IEEE Trans Neural Netw 16(2):498–501
12. Li B, Song S, Li K (2012) Improved conjugate gradient implementation for least squares support vector machines. Pattern Recognit Lett 33(2):121–125
13. Platt J (1998) Sequential minimal optimization: a fast algorithm for training support vector machines
14. Keerthi SS, Shevade SK (2003) Smo algorithm for least-squares svm formulations. Neural Comput 15(2):487–507
15. Fan R-E, Chen P-H, Lin C-J, Joachims T (2005) Working set selection using second order information for training support vector machines. J Mach Learn Res, **6**(12)
16. López J, Suykens JA (2011) First and second order smo algorithms for LS-SVM classifiers. Neural Process Lett 33(1):31–44
17. Shao X, Wu K, Liao B (2013) Single directional smo algorithm for least squares support vector machines. Comput Intell Neurosci **2013**
18. Bo L, Jiao L, Wang L (2007) Working set selection using functional gain for LS-SVM. IEEE Trans Neural Netw 18(5):1541–1544
19. Tavara S (2019) Parallel computing of support vector machines: a survey. ACM Comput Surv (CSUR) 51(6):1–38
20. Cao LJ, Keerthi SS, Ong CJ, Zhang JQ, Periyathamby U, Fu XJ, Lee H (2006) Parallel sequential minimal optimization for the training of support vector machines. IEEE Trans Neural Netw 17(4):1039–1049
21. Zeng Z-Q, Yu H-B, Xu H-R, Xie Y-Q, Gao J (2008) Fast training support vector machines using parallel sequential minimal optimization. In: 2008 3rd International Conference on Intelligent System and Knowledge Engineering, vol. 1, pp. 997–1001. IEEE
22. Cao L, Keerthi SS, Ong CJ, Uvaraj P, Fu XJ, Lee H (2006) Developing parallel sequential minimal optimization for fast training support vector machine. Neurocomputing 70(1–3):93–104

23. Noronha DH, Torquato MF, Fernandes MA (2019) A parallel implementation of sequential minimal optimization on fpga. Microprocess Microsyst 69:138–151
24. Zanghirati G, Zanni L (2003) A parallel solver for large quadratic programs in training support vector machines. Parallel Comput 29(4):535–551
25. Chang P, Bi Z, Feng Y (2014) Parallel smo algorithm implementation based on openmp. In: 2014 IEEE International Conference on System Science and Engineering (ICSSE), pp. 236–240. IEEE
26. Huang S-A, Yang C-H (2019) A hardware-efficient admm-based svm training algorithm for edge computing. arXiv preprint arXiv:1907.09916
27. Cipolla S, Gondzio J (2021) Training large scale SVMS using structured kernel approximations and ADMM. In: PROCEEDINGS OF SIMAI 2020+ 21
28. Torres-Barrán A, Alaíz CM, Dorronsoro JR (2021) Faster svm training via conjugate smo. Pattern Recognit 111:107644
29. López J, Barbero Á, Dorronsoro JR (2011) Momentum acceleration of least–squares support vector machines. In: International Conference on Artificial Neural Networks, pp. 135–142. Springer
30. Torres-Barrán, A., Dorronsoro, J.R.: Nesterov acceleration for the smo algorithm. In: International Conference on Artificial Neural Networks, pp. 243–250 (2016). Springer
31. Chang C-C, Lin C-J (2011) LIBSVM: a library for support vector machines. ACM Trans Intell Syst Technol(TIST) 2(3):1–27
32. Joachims T (1999) Svmlight: Support vector machine. SVM-light support vector machine http://svmlight.joachims.org/, University of Dortmund **19**(4)