STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **16**, August 2026, pp 1447–1465.
Published online in International Academic Press (www.IAPress.org)

An Adaptive Support Method for Training SVM Problems

Louiza Belkhiri-Brahmi [1,*], Belkacem Brahmi [2], Mohand Ouamer Bibi [2]

[1] *LaMOS Research Unit, Department of Computer Science, Faculty of Exact Sciences, University of Bejaia, 06000 Bejaia, Algeria*
[2] *LaMOS Research Unit, Department of Operations Research, University of Bejaia, 06000 Bejaia, Algeria*

Abstract In recent years, extensive research has been conducted on support vector machines (SVMs) and their applications across various scientific fields. Among the various algorithms used for classification and regression in different domains, SVMs are distinguished by their remarkable power and stability. In this context, this study introduces the design and implementation of the Adaptive Support Method-Support Vector Machine (ASM-SVM), a fast training algorithm for SVM classification, a fundamental task in machine learning.
The proposed approach is iterative and determines in a finite number of iterations an optimal or suboptimal solution for the QP (Quadratic Programming) problem. At each iteration of our algorithm, a descent feasible direction and a step size are calculated in order to decrease the value of the objective function and the suboptimality estimate. The major advantage of our algorithm resides in its ability to modify several non-basic variables simultaneously, unlike active-set methods, which only permit changing one variable per iteration. This feature permits reducing the number of iterations and the computational cost that is particularly important for many practical applications. Furthermore, our solution eliminates the need to compute the null-space matrix, making it possible to propose a simple and efficient method suitable for machine implementation. Numerical results obtained on several datasets prove its effectiveness in training both linear and nonlinear SVM classification problems. The ASM-SVM method is designed for easy integration into machine learning software systems.

Keywords Machine learning, classification algorithms, support vector machines, adaptive method, support feasible solution, convergence analysis, training time reduction.

AMS 2010 subject classifications 90C20, 68T09, 90C90, 65K05.

DOI: 10.19139/soic-2310-5070-????

1. Introduction

Machine learning is a highly interdisciplinary field that integrates computer science, cognitive science, optimization, statistics, and various other scientific disciplines. Among its many branches, classification stands as a fundamental supervised learning task, which aims to construct models capable of partitioning data into predefined distinct classes based on observed patterns [51].

The literature abounds with efficient classification methods, including K-Nearest Neighbors (KNN), artificial neural networks, Bayesian networks, decision trees, Support Vector Machines (SVMs), and more recently deep learning architectures [14, 38]. Each of these approaches presents its own strengths and limitations. KNN classifiers are appreciated for their conceptual simplicity and ease of implementation; however, they exhibit high sensitivity to irrelevant features and suffer from poor scalability when confronted with large-scale datasets. Traditional neural networks, while often regarded as universal approximators and successfully deployed across numerous applications, incur significant computational complexity. Their performance critically depends on careful architectural choices—including the number of layers, number of neurons, and the learning algorithm—and they

*Correspondence to: Louiza Belkhiri-Brahmi (Email: louiza.belkhiri@univ-bejaia.dz). LaMOS Research Unit, Department of Computer Science, Faculty of Exact Sciences, University of Bejaia, 06000 Bejaia, Algeria.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

remain highly vulnerable to noise in the training data. Decision trees, in contrast, are more widely adopted in practice due to their faster training times and interpretability, yet they lack the flexibility required to model complex, nonlinear decision boundaries effectively. Deep learning, which extends classical neural networks through multiple hierarchical layers, has achieved remarkable breakthroughs in domains such as computer vision, natural language processing, and speech recognition [38, 29]. However, deep models typically require massive amounts of labeled data, extensive computational resources, and considerable expertise for hyperparameter tuning, which limits their applicability in data-scarce or resource-constrained environments.

Support Vector Machines, introduced by Vapnik and Chervonenkis in the 1960s [53], are powerful machine learning models that have gained significant attention in recent years due to their excellent performance in classification, regression, and many other tasks. Their success is mainly attributed to their solid foundations in statistical learning theory and convex optimization, which ensure global optimality and strong generalization capability. Supported by a rigorous mathematical framework, SVMs have been successfully applied in a wide range of practical domains, including computer vision, IoT systems [1, 33, 3], microelectronic circuit testing [9], intrusion detection [4, 54], image analysis [39, 46], medical diagnosis [5, 49, 37], and pattern recognition [40, 42]. Numerous studies have also reported that SVMs achieve superior performance compared with several other supervised learning techniques [50, 40, 13, 39, 42].

Training an SVM classification problem involves solving a bounded convex quadratic problem (QP) with one linear equality constraint in its dual formulation to find a hyperplane that separates the classes. In practical applications, the Hessian matrix of the quadratic form is large and dense, for which classical optimization methods cannot be applied directly to solve the associated QP. Indeed, such methods become computationally prohibitive in terms of both memory and time complexity. This limitation has motivated the development of specialized optimization strategies tailored to the structure of SVM problems.

Nowadays, there are several efficient methods for solving SVM problems, including decomposition and active-set methods [45, 34, 32, 47, 48, 12], as well as interior-point methods [22, 56, 17]. The basic idea of the two first methods is to solve a smaller QP subproblem at each iteration; the working-set, also called the basis in the simplex method, is then adjusted until the algorithm finds an optimal solution. In the literature, existing methods of this type differ in how they fix the size of the working-set and the strategy adopted to update its elements. For example, the SMO algorithm (Sequential Minimal Optimization) presented in [45] fixes the size of the working-set to only two variables, and consequently, the solution is then calculated analytically without needing to use an optimization solver. The LibSVM solver [15] is the most well-known software implementing this technique, where the strategy of selecting the working-set is based on second-order information [20], which ensures the convergence of the SMO algorithm.

For general decomposition methods [34, 32], the size of the working-set is fixed in advance and is introduced as a parameter by the user. The SVMLight package [34] is one of the state-of-the-art for decomposition methods. In active-set methods [47, 48], the working-set size is variable from an iteration to another and is adjusted by adding or deleting one element of this set. The SVM-QP package [47] implements the dual active set method [28], and singularities can be encountered when an infinite descent direction is calculated. However, the revised simplex method [48] and the primal-dual method [18] allow singularities to be avoided when resolving for the descent direction that implements the null-space technique.

Interior-point methods (IPMs) [22, 56, 17] provide an alternative framework by optimizing all variables simultaneously through barrier functions [57]. At each iteration, a Newton system derived from the Karush-Kuhn-Tucker (KKT) optimality conditions is solved to determine the search direction. IPMs are known for their strong numerical robustness and their polynomial-time complexity, particularly for sparse optimization problems [52]. However, their practical efficiency is often limited by the need to solve large linear systems at every iteration, which leads to significant computational and memory costs and reduces their scalability on large-scale datasets compared with decomposition-based methods.

Adaptive support methods have been developed by Gabasov et al. for solving linear and quadratic problems [23, 25, 26], and have also been extended to solve other problems [23, 24, 11, 9, 36]. These approaches, commonly referred to as support methods, rely on the notion of support, which generalizes the classical concept of a basis in simplex-type algorithms. Unlike the simplex framework, where the basis uniquely determines an extreme-point

solution, the support concept allows more flexibility since a support solution may lie at an extreme point, on a face, or even in the interior of the feasible region. Consequently, the support and the associated solution are not intrinsically linked, which provides additional freedom in constructing feasible iterates and improves numerical robustness. For this reason, adaptive support-based approaches can be regarded as an intermediate framework between active-set methods and interior-point methods.

Recently, adaptive support-based methods have attracted renewed interest and have been successfully applied to several classes of optimization and machine learning problems. Developments include SVM classification problems [18, 12], support vector regression models [2], portfolio optimization and financial decision-making [7, 8], as well as linear fractional programming problems [30]. These recent advances demonstrate the effectiveness of adaptive support methods for handling large-scale, structured, and computationally challenging optimization problems.

In this paper, we present a new computational method for solving the SVM classification problem in its dual formulation. Our approach is iterative and generates a sequence of feasible points for the primal problem. At each iteration, a descent direction is calculated in order to reduce the suboptimality estimate representing the duality gap. Besides, we change the working-basis (support) that ensures the nonsingularity of the new basis matrix. Unlike the classical simplex algorithm [55] and our previous support-based method [12], which update only one nonbasic variable at a time, the proposed ASM-SVM approach allows the simultaneous modification of all nonbasic variables violating the optimality conditions. This strategy significantly accelerates convergence by improving the descent process and reducing the number of iterations required to reach an optimal solution.

The proposed ASM-SVM algorithm is implemented in Matlab. We analyze its performances against popular SVM training algorithms, like the revised simplex method (SVM-RSQP) [48] and the SMO method [45], using several selected datasets from the UCI machine learning repository [19]. For the SMO algorithm, two solvers are considered: (i) SMO-MAT, implemented in MATLAB and invoked via the svmtrain function from the Bioinformatics Toolbox; and (ii) the widely used LibSVM package (version 3.11) [15], written in C++ with a MATLAB interface. Both solvers share the same core techniques, including shrinking, caching, and working set selection strategies [20, 27]. The numerical experiments are encouraging and prove that ASM-SVM achieves better performance than SMO-MAT, LibSVM and SVM-RSQP on on most selected datasets.

The main contributions of this work can be summarized as follows:

- We introduce a novel adaptive support-based framework that dynamically updates the working basis while preserving nonsingularity of the associated matrix.
- We design an iterative algorithm that generates a sequence of feasible primal points and computes descent directions guided by a duality-gap-based optimality measure.
- We establish theoretical convergence results of the proposed ASM-SVM algorithm.
- We introduce a new solution that acts as a bridge between active-set methods and interior-point techniques, effectively combining their complementary strengths.
- We implement the proposed method in MATLAB and perform an extensive experimental evaluation on benchmark datasets, showing that ASM-SVM outperforms classical methods such as SMO-MAT [45], LibSVM [15] and the revised simplex-based SVM solver (SVM-RSQP) [48].

The remainder of this paper is organized as follows. In the next section we present the SVM classification problem, where we give the standard dual formulation of an SVM classification and its primal problem. Section 3 is devoted to the adaptive method for training an SVM problem and we prove its convergence in section 4. Numerical experiments are presented in section 5, where we compare our approach with SMO-MAT, LibSVM and SVM-RSQP on ten benchmarks chosen from the UCI machine learning repository [19]. Section 5 contains some concluded remarks and perspectives of this work.

2. SVM classification problem

We consider a training sample (x_i, y_i), $i \in I = \{1, 2, \ldots, m\}$, where $y_i \in \{-1, +1\}$ is the class of the example $x_i \in \mathbb{R}^p$, m and p are respectively the number of training examples and the number of features. The SVM classification problem consists in solving the following convex quadratic problem:

$$\min_{\alpha} f(\alpha) = \frac{1}{2}\alpha' Q\alpha - e'\alpha, \tag{1a}$$

$$y'\alpha = 0, \tag{1b}$$

$$0 \leq \alpha \leq Ce, \tag{1c}$$

where the m-vector α is the Lagrange multipliers vector associated with the m constraints of the primal problem and e is an m-vector of ones. The $n \times n$ symmetric matrix Q is defined by its elements $q_{ij} = y_i y_j \varphi(x_i, x_j)$, $i, j = 1, \ldots, m$, and is positive semidefinite when the kernel function φ, verifying $\varphi(x_i, x_j) = \phi(x_i)'\phi(x_j)$, fulfills the Mercer's condition [16]. The nonlinear function ϕ projects each input example in a higher-dimensional space, and the constant $C > 0$ is the regularization parameter that controls the trade-off between maximizing the margin and minimizing classification errors. The symbol $(')$ represents the transposition operation and $y = y(I) = (y_i,\, i \in I)$.

The dual problem associated to (1) is the following concave quadratic program:

$$\max_{\alpha, b, s, \xi} -\frac{1}{2}\alpha' Q\alpha - Ce'\xi \tag{2a}$$

$$Q\alpha + by - s + \xi = e, \tag{2b}$$

$$s \geq 0,\ \xi \geq 0, \tag{2c}$$

where s and ξ are m-dimensional vectors representing the slack and surplus variables, respectively. The scalar b is the bias term of the separating hyperplane $w'\phi(x) + b = 0$, where

$$w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$$

is the normal vector recovered from an optimal solution of (1). In the dual formulation, b appears as the Lagrange multiplier associated with the equality constraint in equation (1b). Although expressed in a different form, problem (2) remains fully equivalent to the standard soft-margin primal SVM model (see, e.g., [16, 47]); the reformulation is introduced solely to facilitate the algorithmic developments presented in this paper.

Since the primal problem (1) is convex and its corresponding dual problem (2) is concave, the Karush-Kuhn-Tucker (KKT) first-order optimality conditions are both necessary and sufficient. These conditions characterize an optimal solution α and are formulated as follows [43]:

$$Q\alpha + by - s + \xi = e, \tag{3a}$$

$$y'\alpha = 0, \tag{3b}$$

$$\xi_i(\alpha_i - C) = 0, \quad s_i\alpha_i = 0,\ i = 1, 2, \cdots, m, \tag{3c}$$

$$s_i \geq 0, \quad \xi_i \geq 0, \quad i = 1, 2, \cdots, m, \tag{3d}$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \cdots, m. \tag{3e}$$

Using the notation $\delta = s - \xi$, the stationarity conditions (3a) combined with the complementarity conditions (3c) can be rewritten as follows:

$$\delta = Q\alpha + by - e, \tag{4}$$

such as

$$\begin{cases} \delta_i \geq 0, & \text{if } \alpha_i = 0, \\ \delta_i \leq 0, & \text{if } \alpha_i = C, \\ \delta_i = 0, & \text{if } 0 < \alpha_i < C, \quad i = 1, \ldots, m. \end{cases} \tag{5}$$

From the complementarity conditions (3c), the individual multipliers s_i and ξ_i can be uniquely recovered from δ_i as follows:

- If $\alpha_i = 0$, then $\xi_i = 0$ and $s_i = \delta_i \geq 0$.
- If $\alpha_i = C$, then $s_i = 0$ and $\xi_i = -\delta_i \geq 0$.
- If $0 < \alpha_i < C$, then $s_i = \xi_i = 0$ and $\delta_i = 0$.

More compactly, these relationships can be expressed as $s_i = \max(0, \delta_i)$ and $\xi_i = \max(0, -\delta_i)$, ensuring that the non-negativity constraints (3d) are always satisfied.

Before solving problem (1), let α^* be its optimal solution and b^* its optimal multiplier associated with constraint (1b). Examples with its variables $0 < \alpha_i^* < C$ and $\alpha_i^* = C$ are called *nonbounded* and *bounded support vectors*, respectively. These two types of examples form the *support vectors* that define the separation surface between the two classes. Thus, the decision function for classifying a new example x is defined as follows:

$$\begin{aligned} h(x) &= sign\left(\sum_{i=1}^{m} \alpha_i^* y_i \varphi(x, x_i) + b^*\right) \\ &= sign\left(\sum_{i=1 | 0<\alpha_i^* \leq C}^{m} \alpha_i^* y_i \varphi(x, x_i) + b^*\right). \end{aligned} \tag{6}$$

3. Adaptive Support Method (ASM-SVM) for SVM training

The proposed approach for training an SVM classifier is an extension of the adaptive methods introduced in [23, 25, 26, 11] for solving quadratic programs. It also simplifies the primal-dual method proposed in [18] by avoiding the use of the projected Hessian matrix, which relies on the null-space matrix of the constraints. The algorithm of our method is iterative and generates a sequence of feasible points for both the primal problem and its dual that converge to an optimal solution. This approach can be viewed as an active set approach that uses the information of the primal problem and its dual.

We start this section by providing some definitions and notations related to the proposed method.

Definition 1
Any m-vector α verifying all the constraints (1b)-(1c) is called a feasible solution of Problem (1) and $\delta = Q\alpha + by - e$ is its associated reduced costs vector, where $b \in \mathbb{R}$ is the Lagrange multiplier associated with the equality constraint (1b).

Definition 2
A feasible solution α^* is said to be optimal if it minimizes the objective function f over all the feasible solutions.

Definition 3
A feasible solution α^ϵ is called ϵ-optimal (or suboptimal) if $f(\alpha^\epsilon) - f(\alpha^*) \leq \epsilon$, where α^* is an optimal solution for problem (1) and ϵ is a nonnegative number chosen in advance by the user.

Since problem (1) has one linear equality constraint, so we give here a specific definition of the support concept.

Definition 4 ([12])
A nonempty subset of indices $I_s \subset I$ is called a *support (working-set)* of problem (1), if and only if the associated submatrix $Q_{ss} = Q(I_s, I_s)$ is nonsingular.

Note that for all iterations of our approach, the support set I_s must contain at least one element. The non-support (nonbasic) index set is denoted by $I_n = I \backslash I_s$.

According to the partition of the index set $I = I_s \cup I_n$ of variables α, so the matrix Q and the different vectors are splitted as follows:

$$Q = \begin{pmatrix} Q_{ss} & Q_{sn} \\ Q_{ns} & Q_{nn} \end{pmatrix}, \quad \alpha = \begin{pmatrix} \alpha_s \\ \alpha_n \end{pmatrix}, \quad \delta = \begin{pmatrix} \delta_s \\ \delta_n \end{pmatrix}, \quad y = \begin{pmatrix} y_s \\ y_n \end{pmatrix}, \quad e = \begin{pmatrix} e_s \\ e_n \end{pmatrix}, \quad \text{where } Q_{ss} = Q(I_s, I_s) = (q_{ij},\ i, j \in$$

$I_s)$, $Q_{sn} = Q(I_s, I_n) = (q_{ij},\, i \in I_s,\, j \in I_n) = Q'_{ns}$, $Q_{nn} = Q(I_n, I_n) = (q_{ij},\, i, j \in I_n)$; $\alpha_s = \alpha(I_s) = (\alpha_i,\, i \in I_s)$, $\alpha_n = \alpha(I_n)$; $\delta_s = \delta(I_s)$, $\delta_n = \delta(I_n)$; $y_s = y(I_s)$, $y_n = y(I_n)$; $e_s = e(I_s)$, $e_n = e(I_n)$.

Definition 5
Given a support I_s for problem (1), then the pair $\{\alpha, I_s\}$ is said to be a *support feasible solution (SFS)* of the problem if α verifies relations (1b)-(1c) and $\delta_s = \delta(I_s) = 0$. We say that the SFS $\{\alpha, I_s\}$ is *nondegenerate* if it verifies the following relations:

$$0 < \alpha_i < C,\ \forall i \in I_s.$$

We note that the notion of the SFS generalizes that of the basic feasible solution (BFS) for the simplex method [55], where the nonbasic variables $\alpha_n = (\alpha_i,\, i \in I_n)$ are restricted to take only one of both bounds (0 or C), but they are flexible in the adaptive method, i.e., $0 \le \alpha_i \le C$, $i \in I_n$. Indeed, this novel concept allows the adaptive method to start with any feasible solution for problem (1) that can be an extreme point, a point on the boundary, or an interior point of the feasible region.

3.1. Optimality and suboptimality criteria

The KKT optimality conditions (3) associated with problem (1) can be equivalently reformulated in terms of the support set J_B. The corresponding characterization is derived using arguments analogous to those developed in [23, 25, 26, 11, 12].

Theorem 1 ([23, 25])
Let $\{\alpha, I_s\}$ be a support feasible solution to problem (1). Then, the following relations

$$\begin{cases} \delta_i \ge 0, & \text{if} \quad \alpha_i = 0, \\ \delta_i \le 0, & \text{if} \quad \alpha_i = C, \\ \delta_i = 0, & \text{if} \quad 0 < \alpha_i < C,\ i \in I_n. \end{cases} \tag{7}$$

are sufficient for the optimality of the point α and are also necessary if the SFS $\{\alpha, I_s\}$ is nondegenerate.

It is important to clarify that the optimality conditions stated in Theorem 1 are formulated specifically in terms of the non-support variables, whereas the KKT optimality conditions (5) concern all variables in problem (1).

Definition 6
The following nonnegative number

$$\beta(\alpha, I_s) = \sum_{i \in I_n,\, \delta_i > 0} \delta_i \alpha_i + \sum_{i \in I_n,\, \delta_i < 0} \delta_i (\alpha_i - C) \tag{8}$$

is called the suboptimality estimate of the feasible solution α with respect to the support I_s.

This non-negative quantity measures the maximum deviation between values of the objective function at the optimum α^* and another SFS $\{\alpha, I_s\}$. We now recall the suboptimality criteria.

Theorem 2 ([25, 26])
Let $\{\alpha, I_s\}$ be an SFS for the QP (1), and ϵ a positive number. If we have

$$\beta(\alpha, I_s) \le \epsilon, \tag{9}$$

then α is an ϵ-optimal solution to problem (1).

We have always the inequality

$$f(\alpha) - f(\alpha^0) \le \beta(\alpha, I_s).$$

3.2. An iteration of the proposed method

The aim of the proposed adaptive support method, named ASM-SVM, is to solve the standard formulation of an SVM classifier. Starting with an arbitrary feasible solution, our method iteratively constructs a sequence of support feasible solutions (SFS) that converges globally to an optimal (or potentially suboptimal) solution of problem (1).

At a typical iteration, let $\{\alpha, I_s\}$ be the current SFS and δ the associated reduced costs vector. The first step consists of testing the optimality or suboptimality of the current point. If the optimality conditions given in (7) or the suboptimality criterion in (9) are satisfied, then α is either an optimal or a ϵ-suboptimal solution, and the algorithm terminates accordingly. Otherwise, the algorithm constructs a new solution $\overline{\alpha}$, its corresponding reduced costs vector $\overline{\delta}$, and the new bias term $\overline{b}$, as follows:

$$\overline{\alpha} = \alpha + \theta d, \quad \overline{\delta} = \delta + \theta t, \quad \overline{b} = b + \theta r, \quad \text{with } t = Qd + ry. \tag{10}$$

Here, d and t are n-dimensional vectors representing the improvement directions for the dual and primal problems (1) and (2), respectively. The scalar θ denotes the maximum feasible stepsize along the direction d. The steps of the proposed ASM-SVM algorithm are illustrated by the flowchart shown in Figure 1.

3.2.1. Computation of the adaptive direction: The direction vector $d = (d_s, d_n)'$ is calculated to ensure the feasibility of the new solution $\bar{\alpha}$, and also ensures that $f(\bar{\alpha}) \leq f(\alpha)$. Unlike the classical simplex algorithm [55] and our previous method [12], which update only one nonbasic variable at a time, the proposed ASM-SVM approach allows the simultaneous changing of all nonbasic variables that do not satisfy relations (7). Consequently, the nonbasic components $d_n = (d_i, i \in I_n)$ are computed as follows:

$$d_i = \begin{cases} -\alpha_i, & \text{if } \delta_i > 0, \\ C - \alpha_i, & \text{if } \delta_i < 0, \\ 0, & \text{if } \delta_i = 0,\ i \in I_n. \end{cases} \tag{11}$$

The basic components $d_s = (d_i, i \in I_s)$ are determined by using the following relations:

$$\bar{\delta}_s = \delta_s + \theta t_s = 0 \quad \text{and} \quad y'd = y'_s d_s + y'_n d_n = 0.$$

By the definition of an SFS, we must have $\bar{\delta}_s = 0$ and consequently it results that:

$$t_s = Q_{ss} d_s + Q_{sn} d_n + r y_s = 0.$$

Thus, the subvector d_s and the real r will be obtained by solving the following system of equations:

$$\begin{pmatrix} Q_{ss} & y_s \\ y'_s & 0 \end{pmatrix} \begin{pmatrix} d_s \\ r \end{pmatrix} = \begin{pmatrix} -Q_{sn} d_s \\ -y'_n d_n \end{pmatrix}.$$

Since the basic matrix Q_{ss} is nonsingular by the definition of the support concept, the first block equation yields:

$$d_s = -Q_{ss}^{-1}(Q_{sn} d_n + r y_s), \tag{12}$$

where the scalar r denotes the variation of the bias term, and its value is determined by enforcing the equality constraint

$$y'_s d_s + y'_n d_n = 0.$$

Substituting (12) into the above relation gives

$$-y'_s Q_{ss}^{-1}(Q_{sn} d_n + r\, y_s) + y'_n d_n = 0.$$

Therefore, r is uniquely defined by

$$r = \frac{d'_n (y_n - Q_{ns} Q_{ss}^{-1} y_s)}{y'_s Q_{ss}^{-1} y_s}. \tag{13}$$

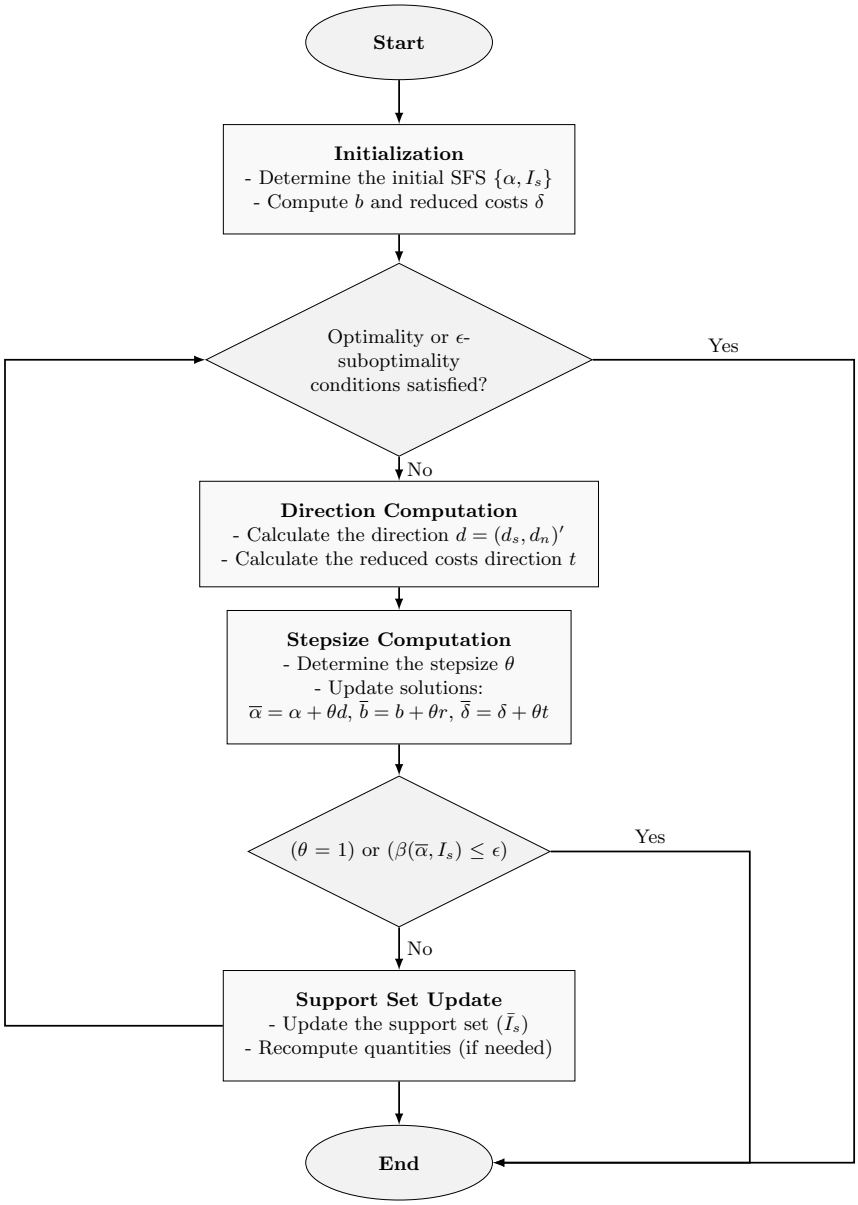

Figure 1. Flowchart of the proposed ASM-SVM algorithm for classification problems

The determination of r, via (13), allows us to calculate the basic component defined in (12), thereby leading to a complete characterization of the direction d.

Now, we can calculate the direction $t = (t_s, t_n)'$ of the reduced costs vector by the following expressions:

$$\begin{cases} t_s = 0, \\ t_n = Q_{ns} d_s + Q_{nn} d_n + r y_n. \end{cases} \tag{14}$$

3.2.2. Computation of the stepsize: The maximal steplength is calculated as follows:

$$\theta = \min\{1, \theta_{i_1}, \sigma_{i_0}\}, \tag{15}$$

where 1 is the value of the stepsize for which all the nonbasic variables are relaxed.

The step θ_{i_1} is calculated to ensure that all basic variables α_s remain feasible:

$$\theta_{i_1} = \min_{i \in I_s} \theta_i, \text{ where } \theta_i = \begin{cases} \frac{-\alpha_i}{d_i}, \text{ if } d_i < 0, \\ \frac{C-\alpha_i}{d_i}, \text{ if } d_i > 0, \\ +\infty, \text{ if } d_i = 0. \end{cases} \tag{16}$$

The stepsize σ_{i_0} is determined in such a way that δ_n and $\bar{\delta}_n$ do not change their signs:

$$\sigma_{i_0} = \min_{i \in I_n} \sigma_i, \text{ with } \sigma_i = \begin{cases} -\frac{\delta_i}{t_i}, \text{ if } \delta_i t_i < 0, \\ +\infty, \text{ otherwise.} \end{cases} \tag{17}$$

In the case $\theta = 1$, we stop at the optimal feasible solution $\bar{\alpha} = \alpha + d$. If not, we calculate the new suboptimality estimate of the SFS $\{\bar{\alpha}, I_s\}$:

$$\beta(\bar{\alpha}, I_s) = (1 - \theta) \left(\beta(\alpha, I_s) - \theta d' Q d\right).$$

This quantity can be further simplified. By using the expression of t given in (14) and the fact that $y'd = 0$, thus, its final expression becomes:

$$\beta(\bar{\alpha}, I_s) = (1 - \theta) \left(\beta(\alpha, I_s) - \theta d'_n t_n\right). \tag{18}$$

If $\beta(\bar{\alpha}, I_s) \leq \epsilon$, then $\bar{\alpha}$ is a suboptimal solution in problem (1), and we stop the algorithm. Otherwise, we will proceed to the final step of the method.

3.2.3. Changing the support: This step consists of obtaining the new basis $\bar{I}_s$ from the precedent one I_s by adding or deleting one element of the support set according to the following cases:

1. For $\theta = \theta_{i_1}, i_1 \in I_s$, we have $\bar{\alpha}_{i_1} = 0$ or $\bar{\alpha}_{i_1} = C$, and we have the two following cases:

   - If $\mid I_s \mid > 1$, then the index i_1 is transferred to $\bar{I}_n$: $\bar{I}_s = I_s \backslash i_1$, $\bar{I}_n = I_n \cup i_1$.
   - If $\mid I_s \mid = 1$, then the index i_1 is exchanged with i_0 as follows: $\bar{I}_s = \{i_0\}$, $\bar{I}_n = I \backslash i_0$. For this case, the novel bias term $\bar{b}$ and the reduced costs vector $\bar{\delta}$ are adjusted so that $\bar{\delta}_s = \bar{\delta}_{i_0} = 0$. These quantities can be updated efficiently without recomputing the gradient at the point $\bar{\alpha}$, i.e., $\nabla f(\bar{\alpha}) = Q\bar{\alpha} - e$:

   $$\begin{cases} \nabla f(\bar{\alpha}) = \bar{\delta} - \bar{b} y, \\ \bar{b} = \nabla f(\bar{\alpha})_{i_0} \, y_{i_0}, \\ \bar{\delta} = \nabla f(\bar{\alpha}) + \bar{b} y. \end{cases} \tag{19}$$

   Here, $\nabla f(\bar{\alpha})_{i_0}$ denotes the i_0-th component of the gradient vector evaluated at $\bar{\alpha}$.

2. For $\theta = \sigma_{i_0}, i_0 \in I_n$, we have $\bar{\delta}_{i_0} = 0$ and do the following index change:

   $$\bar{I}_s = I_s \cup i_0, \qquad \bar{I}_n = I_n \backslash i_0.$$

After this step, the ASM-SVM algorithm is iterated until an optimal or a suboptimal solution is found.

Like any other optimization method, our algorithm requires an initial solution. In our implementation, ASM-SVM can start from the trivial point ($\alpha = 0$) or an interior point, calculated as follows:

$$\alpha_i = \begin{cases} \frac{C}{m^+}, & \text{for } i \in I^+ = \{i : y_i = +1\}, \\ \frac{C}{m^-}, & \text{for } i \in I^- = \{i : y_i = -1\}, \end{cases} \tag{20}$$

where $m^+ = \mid I^+ \mid$ and $m^- = \mid I^- \mid$ are respectively the number of examples in the positive and negative classes. In both cases, the support set is selected independently of the solution, and we put $I_s = \{1\}$. It is clear that the bias b and the reduced costs vector $\delta = (\delta_s, \delta_n)'$ will be obtained as follows:

$$\begin{cases} \delta_s = 0, \qquad b = y_s(e_s - Q(I_s, I)\alpha); \\ \delta_n = Q(I_n, I)\alpha + b y_n - e_n. \end{cases} \tag{21}$$

3.3. ASM-SVM algorithm

The main steps of our ASM-SVM algorithm are given by algorithm 1.

Algorithm 1: ASM-SVM Algorithm for SVM Classification problems

Step 0: Initialization
▷ Start with an initial SFS $\{\alpha, I_s\}$, with $\alpha = 0$ or an interior point from (20), and set $I_s = \{1\}$.
▷ Compute the bias b and the reduced costs vector δ using (21).

Step 1: Optimality/Suboptimality Check
if *optimality conditions* (7) *are satisfied* **then**
  Stop: α is an optimal solution to problem (1);
else if $\beta(\alpha, I_s) \leq \epsilon$ **then**
  Stop: α is an ϵ-suboptimal solution in (1);
end

Step 2: Direction Computation
▷ Compute direction $d = (d_s, d_n)'$ using (11) and (12).
▷ Compute the reduced costs direction $t = (t_s, t_n)'$ using (14);

Step 3: Stepsize Computation
▷ Set $\theta = \min\{1, \theta_{i_1}, \sigma_{i_0}\}$, where θ_{i_1} and σ_{i_0} are calculated from (16) and (17), respectively.
▷ Compute:

$$\overline{\alpha} = \alpha + \theta d, \quad \overline{\delta} = \delta + \theta t, \quad \overline{b} = b + \theta r.$$

Step 4: Suboptimality Update
▷ Calculate the novel suboptimality estimate $\beta(\overline{\alpha}, I_s)$ via (18).
if $\theta = 1$ **then**
  Stop: $\overline{\alpha}$ is an optimal solution.
else if $\beta(\overline{\alpha}, I_s) \leq \epsilon$ **then**
  Stop: $\overline{\alpha}$ is a suboptimal solution.
end

Step 5: Support Set Update
if $\theta = \theta_{i_1},\ i_1 \in I_s$ **then**
  if $|I_s| > 1$ **then**
    $\bar{I}_s = I_s \setminus \{i_1\}, \quad \bar{I}_n = I_n \cup \{i_1\};$
  else
    /⋆ $|I_s| = 1$ ⋆/
    ▷ Set:
    $\bar{I}_s = \{i_0\}, \quad \bar{I}_n = I \setminus \{i_0\}.$;
    ▷ Recompute $\overline{b}$ and $\overline{\delta}$ using (19).
  end
else
  /⋆ $\theta = \sigma_{i_0},\ i_0 \in I_n$ ⋆/
  $\bar{I}_s = I_s \cup \{i_0\}, \quad \bar{I}_n = I_n \setminus \{i_0\};$
end
Go to Step 1 with the updated SFS $\{\overline{\alpha}, \bar{I}_s\}$;

4. Convergence analysis

In this section, we analyze and prove the convergence of the proposed algorithm.

Proposition 1
The vector d calculated by relations (11) and (12) is a descent feasible direction for problem (1).

Proof
At any iteration of the above algorithm, let $\{\alpha, I_s\}$ be an SFS in problem (1) and $\overline{\alpha} = \alpha + \theta d$ another feasible point, such that the m-vector $d = (d_s, d_n)'$ is determined by relations (11) and (12). The stepsize θ is computed with formulas (15), with $\theta \geq 0$.

First, we prove that d is a feasible direction of problem (2). Indeed, we have

$$\begin{aligned} y'd &= y'_s d_s + y'_n d_n \\ &= -y'_s Q_{ss}^{-1} (Q_{sn} d_n + r y_s) + y'_n d_n \\ &= -y'_s Q_{ss}^{-1} Q_{sn} d_n - r y'_s Q_{ss}^{-1} y_s + y'_n d_n. \end{aligned}$$

Since the real r verifies equation (14), this implies that

$$\begin{aligned} r y'_s Q_{ss}^{-1} y_s &= d'_n (y_n - Q_{ns} Q_{ss}^{-1} y_s) \\ &= y'_n d_n - y'_s Q_{ss}^{-1} Q_{sn} d_n. \end{aligned}$$

Finally, we will have

$$\begin{aligned} y'd &= -y'_s Q_{ss}^{-1} Q_{sn} d_n - y'_n d_n + y'_s Q_{ss}^{-1} Q_{sn} d_n + y'_n d_n \\ &= 0, \end{aligned} \tag{22}$$

which means that d is a feasible direction for equation (1). As a consequence, the new point $\bar{\alpha} = \alpha + \theta d$ satisfies the equality constraint of the QP (1). Furthermore, the optimal step $\theta = \min\{1, \theta_{i_1}, \sigma_{i_0}\}$ is calculated such that

$$0 \leq \bar{\alpha_i} \leq C, \ \ \forall i \in I = I_s \cup I_n.$$

Secondly, we prove that d is a descent direction, i.e., $\nabla f(\alpha)' d < 0$, where $\nabla f(\alpha) = Q\alpha - e$ is the gradient of the objective function f at α. The reduced costs vector associated to the point α is given by

$$\delta = Q\alpha + by - e = \nabla f(\alpha) + by \Longrightarrow \nabla f(\alpha) = \delta - by.$$

Since $\delta_s = 0$ by the definition of an SFS and d verifies equation (22), we will have

$$\begin{aligned} \nabla f(\alpha)' d &= \delta' d - by' d \\ &= \sum_{i \in I_n} \delta_i d_i \\ &= -\sum_{i \in I_n : \delta_i > 0} \delta_i \alpha_i - \sum_{i \in I_n : \delta_i < 0} \delta_i (\alpha_i - C) \\ &= -\beta(\alpha, I_s) < 0. \end{aligned}$$

Therefore d is a descent feasible direction.

Moreover, the value of the objective function at the point $\overline{\alpha}$ is expressed by

$$f(\overline{\alpha}) = f(\alpha + \theta d) = f(\alpha) + \theta \nabla f(\alpha)' d + \frac{\theta^2}{2} d' Q d.$$

Since d is a descent feasible direction for problem (1), for any value of the stepsize $\theta > 0$ sufficiently small thus it follows that

$$f(\overline{\alpha}) < f(\alpha). \tag{23}$$

□

Let's now demonstrate the convergence of the adaptive method.

Theorem 3
The algorithm of the adaptive method is finite if, during the process of iterations, a finite number of degenerate supports appear.

Proof
The proof of this theorem is derived from the above proposition. Each ASM-SVM iteration consists in calculating a new feasible solution $\overline{\alpha} = \alpha + \theta d$ from the precedent one α, where d is a decent direction obtained by (11)-(12), and θ is the maximal step size obtained by formula (15).

Indeed, in the case of the nondegeneracy of problem (1), we have $\theta > 0$ and $d \neq 0$. Following the proposition 1, it results that $f(\overline{\alpha}) < f(\alpha)$. In other words, the objective function value decreases at each iteration of the method. So, ASM-SVM algorithm generates a strictly decreasing sequence of feasible solutions. Furthermore, since the number of supports for problem (1) is finite and the algorithm does not visit the same point during its iterations, ASM-SVM starting form any feasible point converges to an optimal or suboptimal solution in a finite number of iterations.

The case of degeneracy occurs only for $\theta = \theta_{i_1} = 0$, $i \in I_s$. For this case, the objective function f remains unchanged and we proceed to construct another support by performing **Step 5** of Algorithm 1. In this case, the entering index is selected according to Bland's rule [6], which induces a lexicographic ordering of degenerate pivots and prevents cycling among support sets. Therefore, degenerate iterations cannot generate an infinite loop over a finite set of supports.

By combining both cases and using the assumption that only a finite number of degenerate supports occur, the total number of iterations (degenerate and non-degenerate) is finite. Hence, the algorithm terminates after a finite number of steps. □

5. Numerical experiments

To assess the effectiveness of the proposed ASM-SVM method, we compare its performance with several well-known state-of-the-art approaches, namely the revised simplex method SVM-RSQP [48], the Sequential Minimal Optimization (SMO-MAT) algorithm implemented in MATLAB [41], and the SMO C++ implementation available in the widely used LIBSVM solver [15].

In this section, we present a numerical comparison between ASM-SVM and these benchmark methods. All experiments were carried out using MATLAB 9.2 (R2017a). The ASM-SVM algorithm was fully implemented by the authors. For the SMO-based approaches, we used both the *svmtrain* function from the MATLAB Bioinformatics Toolbox and the implementation provided in the popular LIBSVM package, which is implemented in C++ and interfaced with MATLAB through MEX functions. For SVM-RSQP, we employed the original MATLAB implementation provided by [48], publicly available at: https://github.com/csentelle/simplexsvm

All experiments were carried out on a personal computer equipped with an Intel Xeon(R) processor running at 2.80 GHz and 32 GB of RAM. The numerical solution accuracy was fixed to 10^{-5} for all methods, while the stopping tolerance was set to $\epsilon = 10^{-3}$. This value corresponds to the default stopping criterion used in LibSVM and also defines the suboptimality tolerance adopted for our ASM-SVM.

Experiments were conducted on ten binary classification datasets (australian, splice, a1a, w1a, Svmguide1, a5a, mushrooms, w5a, phishing and a7a). The majority of these test problems have been taken from the UCI database [19] and available via the internet page of [15]. For each dataset, we have used the linear kernel $\varphi(x_i, x_j) = x_i' x_j$ with the regularization parameter $C = 1$ and the RBF kernel with parameters $\gamma = \frac{1}{m}$ and $C = 1$, such that $\varphi(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$.

The criteria of comparison between the four methods are the number of iterations ($Iters$) obtained at an optimal solution for training each dataset, the CPU time in seconds, the number of support vectors (m_{SV}), the number of bounded support vectors (m_{BSV}), and the training accuracy expressed in percentage. All the features of these experiments were scaled to the range $[0, 1]$.

First, we compare the four methods (ASM-SVM, SVM-RSQP, SMO, and LibSVM) by using the RBF kernel. In this experiment, all methods are initialized from the same starting point ($\alpha = 0$) to ensure a fair and unbiased comparison. For the proposed ASM-SVM algorithm, the initial SFS $\{\alpha, I_s\}$ is constructed with the support set chosen as $I_s = \{1\}$. The comparative results obtained by the four methods are reported in Table 1, while the global trends are illustrated in Figure 2, which summarizes both the number of iterations and CPU time (in logarithmic scale) across all datasets. Best results are highlighted in bold text.

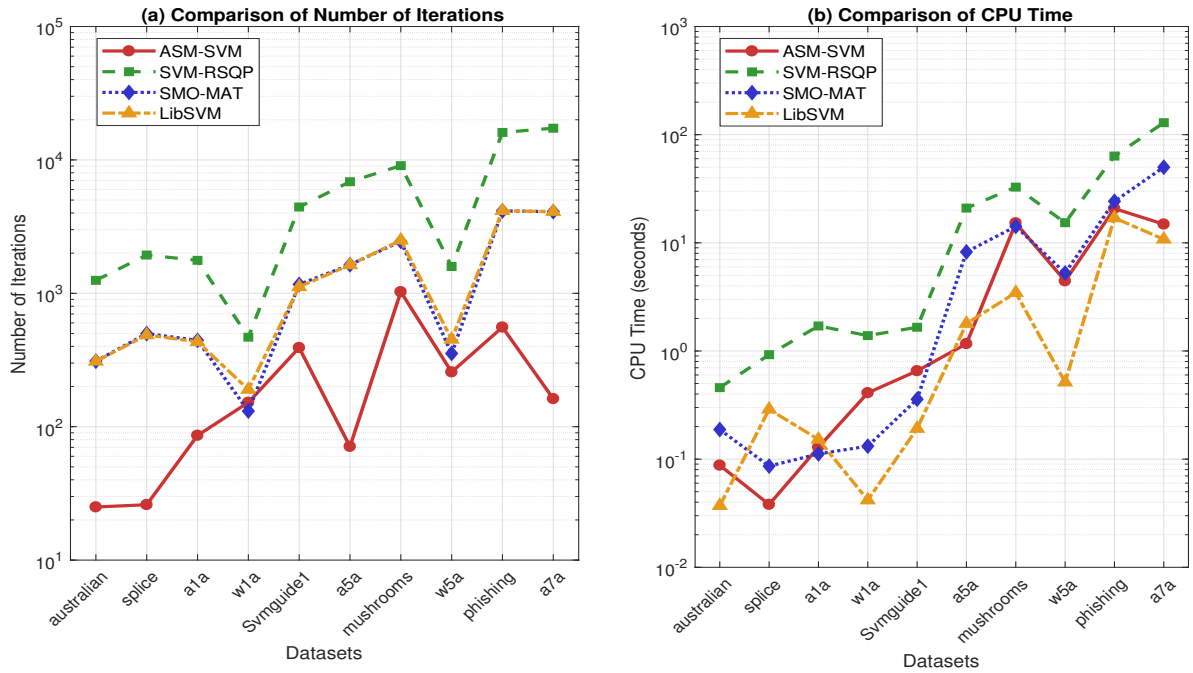

Figure 2. Convergence and computational cost comparison of the four SVM methods using an RBF kernel across 10 benchmark datasets: (a) Number of iterations, and (b) CPU time (seconds).

Based on the numerical results presented in Table 1, ASM-SVM, SVM-RSQP, SMO-MAT, and LibSVM achieve almost identical classification accuracies on all datasets. Moreover, the corresponding numbers of support vectors (m_{SV}) and bounded support vectors (m_{BSV}) are very close for all methods. This observation indicates that the four approaches converge toward comparable optimal solutions and generate nearly identical separating optimal hyperplanes. Hence, the proposed ASM-SVM method preserves the predictive capability and numerical reliability of standard SVM solvers.

Concerning the number of iterations, ASM-SVM consistently outperforms the competing methods on all datasets, as also clearly illustrated in Figure 2(a). In particular, the reduction is highly significant when compared with SVM-RSQP and remains substantial with respect to SMO-MAT and LibSVM solvers. For example, ASM-SVM requires only 25, 26, 71, and 162 iterations for the datasets *australian*, *splice*, *a5a*, and *a7a*, whereas SVM-RSQP requires 1251, 1936, 6864, and 17285 iterations, respectively. This remarkable improvement can be explained by the fact that, at each iteration, ASM-SVM simultaneously updates all nonbasic variables violating the optimality conditions (7), whereas SVM-RSQP updates only one non-basic violating component and the SMO algorithm optimizes only two variables at each iteration. As a consequence, the proposed strategy considerably accelerates convergence, especially for medium and large-scale problems.

With respect to CPU time, the trends reported in Figure 2(b) show that ASM-SVM is generally faster than SVM-RSQP and SMO-MAT on most datasets. The improvement is particularly noticeable for the adult datasets (*a1a*, *a5a*, and *a7a*), where ASM-SVM significantly reduces the computational cost due to its smaller number of iterations. For instance, on the *a5a* dataset, ASM-SVM requires only 1.1663 seconds, while SVM-RSQP and SMO-MAT

Dataset $(m \times p)$	Method	Iters	CPU(s)	m_{SV}	m_{BSV}	Accuracy(%)
australian	ASM-SVM	**25**	0.0878	619	611	77.39
(690×14)	SVM-RSQP	1251	0.4582	615	611	77.39
	SMO-MAT	310	0.1878	615	613	77.39
	LibSVM	309	**0.0371**	615	613	77.39
splice	ASM-SVM	**26**	**0.0381**	968	963	56.50
(1000×60)	SVM-RSQP	1936	0.9251	968	963	56.50
	SMO-MAT	499	0.0862	968	963	56.50
	LibSVM	487	0.2889	968	963	56.50
a1a	ASM-SVM	**86**	0.1288	806	775	75.39
(1605×123)	SVM-RSQP	1767	1.7035	802	780	75.39
	SMO-MAT	444	**0.1118**	800	781	75.39
	LibSVM	433	0.1523	800	782	75.39
w1a	ASM-SVM	152	0.4101	836	91	97.09
(2477×300)	SVM-RSQP	469	1.3872	197	114	97.09
	SMO-MAT	**131**	0.1318	171	123	97.09
	LibSVM	190	**0.0418**	181	117	97.09
Svmguide1	ASM-SVM	**391**	0.6547	2209	2163	64.75
(3089×4)	SVM-RSQP	4437	1.6578	2179	2177	64.75
	SMO-MAT	1166	0.3581	2178	2178	64.75
	LibSVM	1118	**0.1918**	2178	2178	64.75
a5a	ASM-SVM	**71**	**1.1663**	3180	3115	75.54
(6414×123)	SVM-RSQP	6864	20.9586	3152	3125	75.54
	SMO-MAT	1638	8.2448	3144	3132	75.54
	LibSVM	1632	1.7921	3146	3128	75.54
mushrooms	ASM-SVM	**1026**	15.2567	4293	4153	90.04
(8124×112)	SVM-RSQP	9058	32.7499	4219	4193	90.03
	SMO-MAT	2450	14.1935	4212	4200	90.03
	LibSVM	2497	**3.4599**	4218	4195	90.03
w5a	ASM-SVM	**257**	4.4390	3925	423	97.16
(9888×300)	SVM-RSQP	1588	15.3623	609	517	97.16
	SMO-MAT	354	5.2640	585	538	97.16
	LibSVM	451	**0.5143**	600	524	97.16
phishing	ASM-SVM	**557**	20.7334	8036	7976	91.96
(11055×68)	SVM-RSQP	16069	63.4521	8006	7992	91.97
	SMO-MAT	4147	24.2052	8004	7996	91.93
	LibSVM	4159	**16.9907**	8004	7994	91.93
a7a	ASM-SVM	**162**	14.8673	7930	7773	75.66
(16100×123)	SVM-RSQP	17285	129.0542	7849	7827	75.66
	SMO-MAT	4094	50.0320	7839	7833	75.66
	LibSVM	4095	**10.7916**	7841	7831	75.66

Table 1. Comparative results between ASM-SVM, SVM-RSQP, SMO-MAT and LibSVM on ten datasets for the RBF kernel.

require 20.9586 and 8.2448 seconds, respectively. Similarly, on the *phishing* dataset, ASM-SVM substantially decreases the CPU time compared with SVM-RSQP while maintaining the same classification accuracy.

Nevertheless, despite its smaller number of iterations, ASM-SVM is not always the fastest method in terms of execution time. As shown in Figure 2(b) and reported in Table 1, LibSVM achieves the lowest CPU time among all compared methods on several large-scale datasets, including *mushrooms*, *w5a*, and *a7a*.

It's worth noticing that, as shown in Figures 2(a) and 2(b), even when both perform the same number of iterations, the C++ implementation of SMO (LibSVM) is significantly faster than its MATLAB version (SMO-MAT). This well-known disparity results from fundamental differences in programming languages. Indeed, C++ is a compiled language that allows in-place memory updates and benefits from advanced compiler optimizations. In contrast, MATLAB is an interpreted language that relies on copy-on-write memory management. It executes loops slowly and operates within a heavily interactive environment.

Therefore, directly comparing CPU times between ASM-SVM (implemented in MATLAB) and LibSVM (implemented in C++) is inherently unfair, as this comparison confuses algorithmic efficiency with language performance. The SMO-MAT versus LibSVM case proves that for the same algorithm with the same iteration count, MATLAB is slower than C++. In addition, the fact that ASM-SVM (MATLAB) remains competitive with LibSVM (C++) and often outperforms SMO-MAT (Matlab) in terms of iterations number is a strong indicator that ASM-SVM achieves a significantly per-iteration cost. If both methods were implemented in the same compiled language (C++), the iteration count advantage of ASM-SVM would result in clear and substantial superiority in CPU time over LibSVM. Overall, these results demonstrate that ASM-SVM constitutes an efficient and competitive alternative for solving nonlinear SVM problems with the RBF kernel.

In the second part of the numerical experiments, we use the linear kernel with the regularization parameter $C = 1$ in order to compare the four approaches under the same experimental settings as in the previous experiments. Table 2 gives the results of comparison.

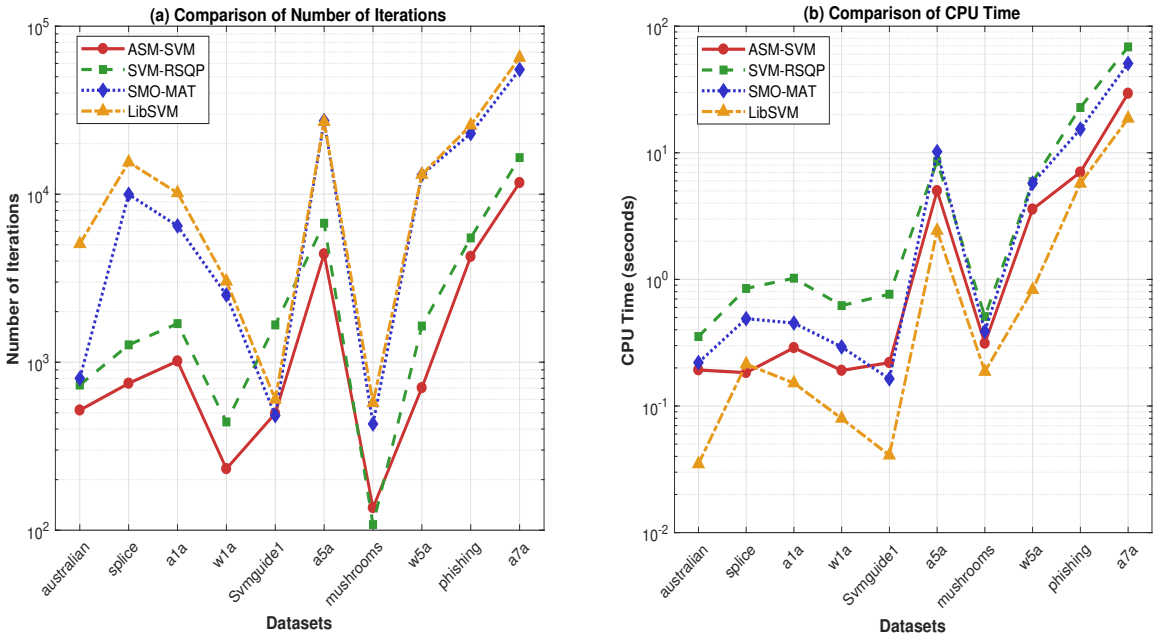

Figure 3. Convergence and computational cost comparison of the four SVM solvers using the linear kernel across 10 benchmark datasets: (a) Number of iterations, and (b) CPU time (seconds).

As shown in Table 2, the classification accuracies achieved by all compared methods are very similar for most of the ten benchmark datasets, which is in line with the results obtained previously for the RBF kernel. The observed differences are marginal (e.g. 84.50% achieved by ASM-SVM versus 84.40% obtained by other solvers on the *splice* dataset), and no method is shown to be significantly more accurate than the others.

Dataset	Method	Iters	CPU(s)	m_{SV}	m_{BSV}	Accuracy(%)
australian	ASM-SVM	**518**	0.1926	219	196	85.51
(690×14)	SVM-RSQP	731	0.3537	211	197	85.51
	SMO-MAT	801	0.2203	236	181	85.51
	LibSVM	5063	**0.0348**	217	194	85.51
splice	ASM-SVM	**748**	**0.1833**	425	364	84.50
(3175×60)	SVM-RSQP	1268	0.8477	425	364	84.40
	SMO-MAT	9974	0.4894	425	364	84.40
	LibSVM	15504	0.2141	426	363	84.40
a1a	ASM-SVM	**1015**	0.2885	587	522	86.23
(1605×123)	SVM-RSQP	1697	1.0224	588	522	86.23
	SMO-MAT	6470	0.4520	589	522	86.23
	LibSVM	10179	**0.1517**	589	522	86.23
w1a	ASM-SVM	**232**	0.1908	170	47	99.27
(2477×300)	SVM-RSQP	441	0.6211	165	47	99.27
	SMO-MAT	2514	0.2928	167	47	99.27
	LibSVM	3028	**0.0797**	166	47	99.27
Svmguide1	ASM-SVM	496	0.2199	782	778	95.11
(3089×4)	SVM-RSQP	1667	0.7635	782	778	95.11
	SMO-MAT	**481**	0.1639	782	778	95.11
	LibSVM	601	**0.0408**	783	778	95.11
a5a	ASM-SVM	**4406**	5.0076	2293	2196	85.02
(6414×123)	SVM-RSQP	6715	8.5539	2286	2198	85.03
	SMO-MAT	27389	10.2106	2287	2197	85.05
	LibSVM	27002	**2.4308**	2290	2195	85.06
mushrooms	ASM-SVM	136	0.3126	335	0	100.00
(8124×112)	SVM-RSQP	**108**	0.5086	58	3	100.00
	SMO-MAT	429	0.3888	236	1	100.00
	LibSVM	573	**0.1870**	239	0	100.00
w5a	ASM-SVM	**704**	3.5819	470	245	98.89
(9888×300)	SVM-RSQP	1642	5.9228	441	258	98.89
	SMO-MAT	13051	5.7438	446	252	98.88
	LibSVM	13074	**0.8277**	441	252	98.89
phishing	ASM-SVM	**4251**	7.0252	1616	1558	93.98
(11055×68)	SVM-RSQP	5492	22.7785	1605	1566	93.97
	SMO-MAT	22971	15.3853	1609	1558	93.98
	LibSVM	25740	**5.7324**	1611	1560	93.97
a7a	ASM-SVM	**11703**	29.4476	5767	5659	84.94
(16100×123)	SVM-RSQP	16507	68.6051	5755	5663	84.90
	SMO-MAT	55195	50.7906	5761	5654	84.92
	LibSVM	65006	**18.6834**	5756	5660	84.89

Table 2. Performance comparison of the four solvers using the linear kernel ($C = 1$).

The convergence behavior is more clearly illustrated in Figure 3(a), which reports the number of iterations required across the different datasets. It can be seen that ASM-SVM consistently requires significantly fewer

iterations than SVM-RSQP, SMO-MAT, and LibSVM. This advantage is particularly pronounced on medium- and large-scale datasets, where classical SMO-type decomposition methods exhibit a substantial increase in the number of iterations. These results confirm the effectiveness of the proposed adaptive active-set strategy, which simultaneously optimizes all nonbasic variables violating the optimality conditions, rather than updating only a limited subset of variables at each iteration.

Nevertheless, two exceptions can be observed on the *svmguide1* and *mushrooms* datasets. On the *svmguide1* dataset, SMO-MAT converges slightly faster than ASM-SVM (481 versus 496 iterations). This behavior can be explained by the fact that, when the total number of support vectors (m_{SV}) is very close to the number of bounded support vectors (m_{BSV}), SMO-type methods benefit from a favorable solution structure where most variables are fixed at their upper bounds ($\alpha_i = C$), making them highly competitive with active-set approaches.

On the *mushrooms* dataset, SVM-RSQP requires fewer iterations than ASM-SVM (108 versus 136 iterations). However, this advantage in iteration count is mitigated by a significantly higher computational cost per iteration. Indeed, SVM-RSQP relies on the reduced Hessian matrix at every iteration to compute its search direction, whereas ASM-SVM avoids this costly operation through its adaptive updating strategy. As a result, SVM-RSQP requires (0.5086 *s*) of CPU time compared with only (0.3126 *s*) for ASM-SVM, making it approximately ($1.6\times$) slower. Consequently, these isolated cases do not affect the overall convergence advantage of ASM-SVM solver.

Regarding computational efficiency, Figure 3(b) compares the CPU time of all methods. The results show that ASM-SVM achieves competitive execution times with respect to SMO-MAT and SVM-RSQP, while maintaining a clear advantage in convergence efficiency. However, LibSVM remains the fastest method in several cases due to its highly optimized implementation as explained before. Despite this, ASM-SVM demonstrates a favorable balance between iteration reduction and computational cost, particularly on larger datasets where convergence behavior dominates performance.

Overall, numerical results consistently demonstrate that ASM-SVM significantly reduces the number of iterations while maintaining competitive CPU time and identical classification accuracy. These results confirm that the proposed method is an efficient and robust alternative for solving large-scale SVM problems.

6. Conclusion

In this paper, we have proposed a new adaptive method for training SVM problems and proved its convergence. This latter is an enhancement of the primal-dual method [18] for training SVM problems. Our approach is based on the concept of the support, which generalizes that of the basis in the simplex method and uses the suboptimality estimate as a stopping criterion. At each iteration, the diminution of the suboptimality estimate is ensured and also keeps the nonsingularity of the support matrix that is the case for the revised simplex method [48]. In addition, ASM-SVM allows to change all nonbasic components that is not the case for the simplex method that changes only the most violated nonbasic variable. This particularity permits to speedup the convergence of the adaptive method.

Numerical experiments on benchmark datasets demonstrate that the proposed method consistently reduces the number of iterations while achieving competitive CPU time and identical classification accuracy compared to state-of-the-art solvers. These results highlight the efficiency and robustness of ASM-SVM for large-scale SVM training.

As future work, we plan to enhance the scalability of the ASM-SVM algorithm by incorporating caching and chunking strategies commonly used in decomposition and active-set methods [34, 45]. Furthermore, we intend to develop a `C++` implementation of ASM-SVM to enable a fair and efficient comparison with widely used SVM solvers, such as `LibSVM` [15], `SVMLight` [34], and `SVM-QP` [47]. In addition, we aim to extend the application of ASM-SVM to other machine learning tasks such as regression, anomaly detection, and multi-class classification.

Finally, a promising direction for future research would be to extend the proposed ASM-SVM framework to other SVM formulations, particularly the primal formulation, and to evaluate its performance against state-of-the-art large-scale optimization methods such as Pegasos [44], stochastic gradient descent (SGD) [10], and dual coordinate descent methods [31, 21].

We also intend to investigate a direct multi-class extension of the adaptive support framework, either via one-versus-one or one-versus-rest strategies, and to report the resulting trade-off between computational cost and classification accuracy. These enhancements will be accompanied by a standalone C++ implementation to ensure reproducibility and scalability in production environments.

Acknowledgement. We would like to thank the editor and the two anonymous reviewers for their valuable and constructive comments, which helped us to improve this paper.

REFERENCES

1. A. Alqarni, *A support vector machine (SVM) model for privacy recommending data processing model (PRDPM) in internet of vehicles*, Computers, Materials & Continua, vol. 82, no. 1, pp. 389-406, 2025.
2. S. Azib, and B. Brahmi, *A working set algorithm for support vector regression problems*, Journal of Numerical Analysis and Approximation Theory, vol. 54, no. 2, pp. 211-228, 2025.
3. H. Balogun, H. Alaka and C. N. Egwim, *Boruta-grid-search least square support vector machine for NO_2 pollution prediction using big data analytics and IoT emission sensors*, Applied Computing and Informatics, vol. 21, no. 1/2, pp. 101-113, 2025.
4. B. S. Bhati and C. S. Rai, *Analysis of support vector machine-based intrusion detection techniques*, Arabian Journal for Science and Engineering, vol. 45, no. 4, pp. 2371-2383, 2020.
5. A. Bilal, A. Imran, T. I. Baig, X. Liu, E. Abouel Nasr and H. Long, *Breast cancer diagnosis using support vector machine optimized by improved quantum inspired grey wolf optimization*, Scientific Reports, vol. 14, no. 1, 2024.
6. R. G. Bland, *New finite pivoting rules for the simplex method*, Mathematics of Operations Research, vol. 2, no. 2, pp. 103-107, 1977.
7. S. Boudjelda, and B. Brahmi, *Parametric Support approach for solving Mean-Variance problem under general constraints*, Statistics, Optimization & Information Computing, vol. 13, no. 1, pp. 189-208, 2025.
8. S. Boudjelda, and B. Brahmi, *Parametric direct support method for solving the bi-objective portfolio optimization problem*, Annals of Operations Research, vol. 356, no. 2-3, pp. 697-725, 2026.
9. A. Bounceur, S. Djemai, B. Brahmi, M. O. Bibi, and R. Euler, *A classification approach for an accurate Analog/RF BIST evaluation based on the process parameters*, Journal of Electronic Testing, vol. 34, no. 3, pp. 321-335, 2018.
10. L. Bottou, *Large-scale machine learning with stochastic gradient descent*, in Proceedings of COMPSTAT, pp. 177-186, 2010.
11. B. Brahmi, and M. O. Bibi, *Dual support method for solving convex quadratic programs*, Optimization, vol. 59, no. 6, pp. 851-872, 2010.
12. B. Brahmi, *An efficient primal simplex method for solving large-scale support vector machines*, Neurocomputing, vol. 599, article 128109, 2024.
13. J. Cervantes, F. G. Lamont, A. López-Chau, L. R. Mazahua and J. S. Ruíz, *Data selection based on decision tree for SVM classification on large data sets*, Applied Soft Computing, vol. 37, pp. 787-798, 2015.
14. J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua and A. Lopez, *A comprehensive survey on support vector machine classification: Applications, challenges and trends*, Neurocomputing, vol. 408, pp. 189-215, 2020.
15. C. C. Chang, and C. J. Lin, *LibSVM: A library for support vector machines*, ACM Transactions on Intelligent Systems and Technology, vol. 2, no. 3, article 27, 2011.
16. N. Cristianini, and J. Shawe-Taylor, *Introduction to support vector machines and other kernel-based learning methods*, Cambridge University Press, 2000.
17. X. Ding, S. Jin, M. Lei, and F. Yang, *A predictor-corrector affine scaling method to train optimized extreme learning machine*, Journal of the Franklin Institute, vol. 359, no. 2, pp. 1713-1731, 2022.
18. S. Djemai, B. Brahmi, and M. O. Bibi, *A primal-dual method for SVM training*, Neurocomputing, vol. 211, pp. 34-40, 2016.
19. D. Dua, and C. Graff, *UCI Machine Learning Repository*, University of California, Irvine, (2017). Available at: http://archive.ics.uci.edu/ml.
20. R. E. Fan, P. H. Chin, and C. J. Lin, *Working set selection using second order information for training support vector machines*, Journal of Machine Learning Research, vol. 6, pp. 1889-1918, 2005.
21. R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang and C.-J. Lin, *LIBLINEAR: A library for large linear classification*, Journal of Machine Learning Research, vol. 9, pp. 1871-1874, 2008.
22. M. C. Ferris, and T. S. Munson, *Interior-point methods for massive support vector machines*, SIAM Journal on Optimization, vol. 13, no. 3, pp. 783-804, 2002.
23. R. Gabasov, F. M. Kirillova, and V. M. Raketsky, *On methods for solving the general problem of convex quadratic programming*, Soviet Mathematics Doklady, vol. 23, pp. 653-657, 1981.
24. R. Gabasov, F. M. Kirillova, and A. Tyatyushkin, *Constructive methods of optimization. Vol. 1: Linear problems*, University Press, Minsk, 1984.
25. R. Gabasov, F. M. Kirillova, and O. I. Kostyukova, *Solution of linear quadratic extremal problems*, Soviet Mathematics Doklady, vol. 31, pp. 99-103, 1985.
26. R. Gabasov, F. M. Kirillova, V. M. Raketsky, and O. I. Kostyukova, *Constructive methods of optimization. Vol. 4: Convex problems*, University Press, Minsk, 1987.
27. T. Glasmachers, C. Igel, K. P. Bennett, and E. Parrado-Hernández, *Maximum-gain working set selection for SVMs*, Journal of Machine Learning Research, vol. 7, pp. 1437-1466, 2006.
28. D. Goldfarb, and A. U. Idnani, *A numerically stable dual method for solving strictly convex quadratic problems*, Mathematical Programming, vol. 27, pp. 1-33, 1983.

29. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016. Available at: http://www.deeplearningbook.org.
30. M. A. Hakmi, M. Bentobache and M. O. Bibi, *A Hybrid Direction Method for Linear Fractional Programming*, Statistics, Optimization & Information Computing, vol. 13, no. 3, pp. 922-947, 2025.
31. C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi and S. Sundararajan, *A dual coordinate descent method for large-scale linear SVM*, in Proceedings of the 25th International Conference on Machine Learning (ICML), pp. 408-415, ACM, 2008.
32. C. W. Hsu, and C. J. Lin, *A simple decomposition method for support vector machines*, Machine Learning, vol. 46, pp. 291-314, 2002.
33. C. Ioannou and V. Vassiliou, *Network attack classification in IoT using support vector machines*, Journal of Sensor and Actuator Networks, vol. 10, no. 3, p. 58, 2021.
34. T. Joachims, *Making large-scale SVM learning practical*, In: B. Schölkopf, C. J. Burges, and A. J. Smola (Eds.), *Advances in kernel methods: Support vector learning*, pp. 169-184, MIT Press, 1998.
35. N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica, vol. 4, no. 4, pp. 373-395, 1984.
36. N. Khimoum, and M. O. Bibi, *Primal-dual method for solving a linear-quadratic multi-input optimal control problem*, Optimization Letters, vol. 14, no. 3, pp. 653-669, 2020.
37. G. Khyathi, K. P. Indumathi, A. Jumana Hasin, M. Lisa Flavin Jency, G. Krishnaprakash and F. J. M. Lisa, *Support vector machines: a literature review on their application in analyzing mass data for public health*, Cureus, vol. 17, no. 1, 2025.
38. Y. LeCun, Y. Bengio and G. Hinton, *Deep learning*, Nature, vol. 521, no. 7553, pp. 436-444, 2015.
39. X. Liang, L. Zhu and D. S. Huang, *Multi-task ranking SVM for image cosegmentation*, Neurocomputing, vol. 247, pp. 126-136, 2017.
40. Y. Ma and G. Guo (eds.), *Support vector machines applications*, vol. 649, Springer, New York, 2014.
41. The MathWorks, Inc., *MATLAB and Bioinformatics Toolbox: User's Guide*, Version 2017a, Natick, Massachusetts, United States, 2017.
42. V. A. Naik and A. A. Desai, *Online handwritten Gujarati character recognition using SVM, MLP, and K-NN*, in 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1-6, IEEE, 2017.
43. J. Nocedal, and S. J. Wright, *Numerical optimization*, Springer-Verlag, 1999.
44. S. Shalev-Shwartz, Y. Singer and N. Srebro, *Pegasos: Primal estimated sub-gradient solver for SVM*, in Proceedings of ICML, pp. 807-814, 2007.
45. J. C. Platt, *Fast training of support vector machines using sequential minimal optimization*, In: B. Schölkopf, C. Burges, and A. Smola (Eds.), *Advances in kernel methods: Support vector learning*, pp. 185-208, MIT Press, 1999.
46. B. M. Rashed and N. Popescu, *Medical image-based diagnosis using a hybrid adaptive neuro-fuzzy inferences system (ANFIS) optimized by GA with a deep network model for features extraction*, Mathematics, vol. 12, no. 5, p. 633, 2024.
47. K. Scheinberg, *An efficient implementation of an active set method for SVM*, Journal of Machine Learning Research, vol. 7, pp. 2237-2257, 2006.
48. C. Sentelle, G. C. Anagnostopoulos, and M. Georgiopoulos, *Efficient revised simplex method for SVM training*, IEEE Transactions on Neural Networks, vol. 22, no. 10, pp. 1650-1661, 2011.
49. L. Shen, H. Chen, Z. Yu, W. Kang, B. Zhang, H. Li, and D. Liu, *Evolving support vector machines using fruit fly optimization for medical data classification*, Knowledge-Based Systems, vol. 96, pp. 61-75, 2016.
50. B. Y. Sun, D. S. Huang and H. T. Fang, *Lidar signal denoising using least-squares support vector machine*, IEEE Signal processing letters, vol. 12, no. 2, pp. 101-104, 2005.
51. P.-N. Tan, M. Steinbach and V. Kumar, *Introduction to Data Mining*, Pearson, 2005.
52. K.-T. D. Thi, *Quadratic programming and quadratically constrained quadratic programming: theory, algorithms, and applications*, HPU2 Journal of Science: Natural Sciences and Technology, vol. 4, no. 3, pp. 80-96, 2025.
53. V. N. Vapnik and A. Y. Chervonenkis, *On the uniform convergence of relative frequencies of events to their probabilities*, in Measures of complexity: festschrift for alexey chervonenkis, pp. 11-30, Springer, Cham, 2015.
54. P. Waghmode, M. Kanumuri, H. El-Ocla and T. Boyle, *Intrusion detection system based on machine learning using least square support vector machine*, Scientific Reports, vol. 15, no. 1, article 12066, 2025.
55. P. Wolfe, *The simplex method for quadratic programming*, Econometrica, vol. 27, pp. 382-398, 1959.
56. K. Woodsend and J. Gondzio, *Exploiting separability in large-scale linear support vector machine training*, Computational Optimization and Applications, vol. 49, no. 2, pp. 241-269, 2009.
57. S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, 1997.