Journal of Optimization Theory and Applications
https://doi.org/10.1007/s10957-022-02103-1

New Interior-Point Approach for One- and Two-Class Linear Support Vector Machines Using Multiple Variable Splitting

Jordi Castro[1]

Received: 17 January 2022 / Accepted: 24 August 2022

Abstract
Multiple variable splitting is a general technique for decomposing problems by using copies of variables and additional linking constraints that equate their values. The resulting large optimization problem can be solved with a specialized interior-point method that exploits the problem structure and computes the Newton direction with a combination of direct and iterative solvers (i.e. Cholesky factorizations and preconditioned conjugate gradients for linear systems related to, respectively, subproblems and new linking constraints). The present work applies this method to solving real-world binary classification and novelty (or outlier) detection problems by means of, respectively, two-class and one-class linear support vector machines (SVMs). Unlike previous interior-point approaches for SVMs, which were practical only with low-dimensional points, the new proposal can also deal with high-dimensional data. The new method is compared with state-of-the-art solvers for SVMs that are based on either interior-point algorithms (such as SVM-OOPS) or specific algorithms developed by the machine learning community (such as LIBSVM and LIBLINEAR). The computational results show that, for two-class SVMs, the new proposal is competitive not only against previous interior-point methods—and much more efficient than they are with high-dimensional data—but also against LIBSVM, whereas LIBLINEAR generally outperformed the proposal. For one-class SVMs, the new method consistently outperformed all other approaches, in terms of either solution time or solution quality.

Keywords Interior-point methods · Support vector classifier · One-class support vector machine · Multiple variable Splitting · Large-scale optimization

Mathematics Subject Classification 90C51 · 90C20 · 90C90 · 62H30

Communicated by Goran Lesaja.

✉ Jordi Castro
jordi.castro@upc.edu

[1] Department of Statistics and Operations Research, Universitat Politècnica de Catalunya, UPC, Barcelona, Catalonia

Published online: 29 September 2022

1 Introduction

Machine learning applications require the solution of a—usually large—optimization problem [2]. In the case of support vector machines (SVM, one of the preferred tools in machine learning), the optimization problem to be solved is convex and quadratic [10]. SVMs can be used for either binary classification or novelty detection. When used for binary classification, they are referred to as a support vector classifier or two-class SVM [9]; for novelty (or outlier) detection, they are called one-class SVM [8, 20]. Although in recent years they have been replaced by *neural networks* in some applications (e.g. for image detection and classification), SVMs are still one of the preferred techniques for text classification [2].

In this work, we present a new approach for solving real-world two-class and one-class SVMs. It is based on reformulating the SVM problem by decomposing it into smaller SVMs and using linking constraints to equate the values of the split variables. The resulting optimization problem has a primal block-angular structure which can be efficiently solved using the specialized interior-point method (IPM) of [3–6]. The extensive computational experience in Sect. 4 shows that the new approach can be competitive against state-of-the-art methods for two-class SVM and that it outperformed all of them for one-class SVM.

Briefly, SVMs attempt to find a hyperplane separating two classes of multidimensional points (two-class SVM), or points with some distribution from a set of outliers (one-class SVM). For either one-class or two-class SVMs, we are given a set of p d-dimensional points $a_i \in \mathbb{R}^d$, $i = 1, \ldots, p$. Each point could be related to some item, and the d components of the point would be related to variables (named *features* in machine learning jargon) for that item. In two-class SVMs, we also have a vector $y \in \mathbb{R}^p$ of labels $y_i \in \{+1, -1\}$, $i = 1, \ldots, p$, indicating whether point i belongs to class "+1" or class "−1". In some applications, points a_i need to be previously transformed by function $\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$, especially if the two classes of points cannot be correctly separated by a hyperplane. When dimension d is high, such a transformation is usually not needed, since a good separation hyperplane can be found. That is, $\phi(x) = x$, and we refer to this problem as the *linear SVM*. In this work, we focus on linear SVMs.

1.1 The Two-Class SVM Optimization Problem

For the two-class SVM (or support vector classifier), we compute a plane $w^\top x + \gamma = 0$, $w \in \mathbb{R}^d$, $\gamma \in \mathbb{R}$, such that points a_i with $y_i = +1$ should be in the half-plane $w^\top x + \gamma \geq 1$, and those points with label $y_i = -1$ should be in the half-plane $w^\top x + \gamma \leq -1$. Slack variables $s \in \mathbb{R}^p$ are introduced to account for possible misclassification errors if the data points are not linearly separable. At the same time, we also attempt to maximize the distance between the parallel planes $w^\top x + \gamma = 1$ and $w^\top x + \gamma = -1$, such that the two classes of points are far enough from each other. This distance, named *separation margin*, is $2/\|w\|$ [10]. Using $A \in \mathbb{R}^{p \times d}$ to define the matrix storing row-wise the p d-dimensional points a_i, $i = 1, \ldots, p$, and $Y = \text{diag}(y)$ to define the diagonal matrix made from the vector of labels y, the primal formulation of the two-class SVM problem is

$$\min_{w,\gamma,s} \quad \frac{1}{2} w^\top w + \nu e^\top s \tag{1a}$$

$$\text{subject to} \quad Y(Aw + \gamma e) + s \geq e \tag{1b}$$

$$s \geq 0, \tag{1c}$$

where $e \in \mathbb{R}^p$ is a vector of 1s, and ν is a fixed parameter to balance the two opposite terms of the objective function: the first quadratic term maximizes the separation margin, and the second term minimizes the misclassification errors.

Defining the vectors of Lagrange multipliers $\lambda \in \mathbb{R}^p$ and $\mu \in \mathbb{R}^p$, respectively, for constraints (1b) and (1c), the Lagrangian function of (1) is

$$L(w, \gamma, s, \lambda, \mu) = \frac{1}{2} w^\top w + \nu e^\top s - \lambda^\top (Y(Aw + \gamma e) + s - e) - \mu^\top s, \tag{2}$$

and the Wolfe dual of (1) becomes

$$\max_{w,\gamma,s,\lambda,\mu} \quad L(w, \gamma, s, \lambda, \mu) \tag{3a}$$

$$\text{subject to} \quad \nabla_w L(\cdot) = w - (\lambda^\top Y A)^\top = 0 \tag{3b}$$

$$\nabla_\gamma L(\cdot) = \lambda^\top y = 0 \tag{3c}$$

$$\nabla_s L(\cdot) = \nu e - \lambda - \mu = 0 \tag{3d}$$

$$\lambda \geq 0, \quad \mu \geq 0. \tag{3e}$$

Using relations (3b)–(3e) in (3a), we obtain the dual problem (in minimization form)

$$\min_{\lambda} \quad \frac{1}{2} \lambda^\top Y A A^\top Y \lambda - \lambda^\top e \tag{4a}$$

$$\text{subject to} \quad \lambda^\top y = 0 \tag{4b}$$

$$0 \leq \lambda \leq \nu e. \tag{4c}$$

The dual (4) is a convex quadratic optimization problem with only one linear constraint and simple bounds. The linear constraint (4b) comes from $\nabla_\gamma L(\cdot)$. Therefore, if a linear (instead of an affine) separation plane $w^\top x = 0$ is considered in the primal formulation (that is, without the γ term), the dual is only defined by (4a) and (4c). Such a problem can be effectively dealt with by gradient and coordinate descent methods [23]. This fact is exploited by some of the most popular and efficient packages in machine learning (such as LIBLINEAR [11]). We note, however, that both problems are slightly different, since they compute either a linear or an affine separation plane.

1.2 The One-Class SVM Optimization Problem

The purpose of the one-class SVM problem (introduced in [20]) is to find a hyperplane $w^\top x - \gamma = 0$, $w \in \mathbb{R}^d$, $\gamma \in \mathbb{R}$, such that points in the half-plane $w^\top x - \gamma \geq 0$ are considered as belonging to the same distribution, and the separation margin with

respect to the origin is maximized. Points that are not in the previous half-plane are considered outliers. Defining, as in the two-class SVM problem, the matrix $A \in \mathbb{R}^{p\times d}$ whose row i contains point a_i, $i = 1, \ldots, p$, the primal formulation of the one-class SVM problem is

$$\min_{w,\gamma,s} \quad \frac{1}{2} w^\top w - \gamma + \frac{1}{\nu p} e^\top s \tag{5a}$$

$$\text{subject to} \quad Aw - \gamma e + s \geq 0 \tag{5b}$$

$$s \geq 0, \tag{5c}$$

where the positive components of s in the optimal solution would be associated with outliers, and $\nu \in [0, 1]$ is a fixed parameter. It was shown in [20] that ν is an upper bound on the fraction of detected outliers in the optimal solution.

As in the two-class SVM problem, we can use Wolfe duality to compute the dual of (5), thereby obtaining:

$$\min_{\lambda} \quad \frac{1}{2} \lambda^\top A A^\top \lambda \tag{6a}$$

$$\text{subject to} \quad \lambda^\top e = 1 \tag{6b}$$

$$0 \leq \lambda \leq \frac{1}{\nu p} e. \tag{6c}$$

One significant difference with respect to the two-class SVM problem is that the linear constraint (6b) cannot be avoided by removing γ from the primal formulation (5) (that is, by computing a linear instead of an affine plane): if γ was removed, problem (5) would have the trivial and useless solution $w^* = 0$, $s^* = 0$. As will be shown in the computational results of Sect. 4, this fact has far reaching implications for methods that solve (6) by means of coordinate gradient descent [8], as they may provide a poor quality solution that is far from the optimal one.

1.3 Alternative Approaches for SVMs

Several approaches have been developed for solving the two-class SVM problem by using either (1) or (4). We will avoid giving an extensive list and will focus instead on only those based on IPMs (like ours) and those implemented in the current state-of-the-art packages for SVMs that will be used in the computational results of this work.

Since (1) and (4) are convex quadratic linearly constrained optimization problems, they can be solved by a general solver implementing an IPM. However, when using either the primal or dual formulation, computing the Newton direction would mean solving a linear system involving matrix $A\Theta A^\top \in \mathbb{R}^{p\times p}$ (where Θ is some diagonal scaling matrix that is different for each IPM iteration). For datasets with a large number of points p, the Cholesky factorization can be prohibitive because matrices A are usually quite dense. However, for low-rank matrices A which involve many points and just a few variables (that is, $p \gg d$), a few very efficient approaches have been

devised. The first one was that of [12], who considered the dual problem (4) and solved the Newton system by applying the Sherman–Morrison–Woodbury (SMW) formula. In [12], the authors solved problems with millions of points but only $d = 35$ features. A similar approach was used in [13] for the solution of smaller (up to 68,000 points, and less than 1000 features) but realistic datasets. The product form Cholesky factorization introduced in [14] for IPMs with dense columns was applied in [15] for solving the dual SVM formulation. This approach was shown to have a better numerical performance than those based on the SMW formula, but no results for real SVM instances were reported in [15]. State-of-the-art IPM solvers including efficient strategies for dealing with dense columns (such as CPLEX) can also be used for solving the primal formulation (1). Indeed, the computational results of Sect. 4 will show that CPLEX 20.1 is competitive against specialized packages for both one-class and two-class SVMs when d is small.

More recently, [21] suggested a separable reformulation of (4) by introducing the extra free variables u. The resulting problem

$$\begin{aligned} \min_{\lambda} \ & \tfrac{1}{2} u^\top u - \lambda^\top e \\ & \lambda^\top y = 0 \\ & A^\top Y \lambda = u \\ & 0 \le \lambda \le \nu e, \quad u \text{ free}, \end{aligned}$$

was efficiently solved when A is low-rank. This approach was implemented in the SVM-OOPS package, and [21] extensively tested it against state-of-the-art machine learning packages for SVMs, showing competitive results (but only for problems with a few features). SVM-OOPS can be considered one of the most efficient specialized IPM approaches for linear SVMs when d is small, and it will be one of the packages considered in Sect. 4 for the computational results (but only for classification, since it does not deal with one-class SVMs).

The two likely best packages for SVMs in the machine learning community are LIBSVM [7] and LIBLINEAR [11]. Both will be used for comparison purposes in the computational results of Sect. 4. LIBSVM solves the duals (4) and (6) (and can thus be used for both two-class and one-class SVMs) without removing the linear constraint (that is, it considers the γ term of the primal formulation). It applies a gradient descent approach combined with a special active set constraint technique named sequential minimal optimization (SMO) [19], where all but two components of λ are fixed, and each iteration deals only with a two-dimensional subproblem. Each iteration of SMO is very fast, but convergence can be slow. As will be shown in Sect. 4, LIBSVM is generally not competitive against the other methods. On the other hand, it is the only one that can efficiently deal with nonlinear SVMs—that is, when $\phi(x)$ is a general transformation function.

LIBLINEAR [11] is considered the fastest package for linear SVMs. For two-class SVMs, it solves either the primal or the dual formulation without the γ variable. For the primal formulation, it considers the following approximate unconstrained reformulation:

$$\min_{w} \frac{1}{2} w^\top w + \nu \sum_{i=1}^{p} \max(0, 1 - y_i w^\top a_i)^2.$$

The nondifferentiable max() term (known as *hinge loss function* in the machine learning field) must be squared to avoid differentiability issues with the first derivative (this term, however, has no second derivative at 0). This unconstrained problem is solved by a trust-region Newton method based on conjugate gradients while further using a Hessian perturbation to deal with the second derivatives at 0. Due to the lack of γ, LIBLINEAR solves the dual formulation (4) without the linear constraint:

$$\begin{aligned} \min_{\lambda} \ & \tfrac{1}{2} \lambda^\top Y A A^\top Y \lambda - \lambda^\top e \\ & 0 \leq \lambda \leq \nu e. \end{aligned} \tag{7}$$

This problem is solved using a coordinate gradient descent algorithm. For one-class SVM, LIBLINEAR solves only the exact dual (6) (which includes the linear constraint) by means of a coordinate descent algorithm [8]. Alternative approaches also considered nondifferentiable formulations [1], but resulted less efficient than that implemented in LIBLINEAR.

The rest of the paper is organized as follows: Section 2 introduces the multiple variable splitting reformulation considered in this work for linear SVMs. Section 3 presents the specialized IPM that will be used for the efficient solution of the multiple variable splitting reformulation of the SVM problem. Finally, computational results for one-class and two-class SVMs using real datasets will be provided in Sect. 4, showing the efficiency and competitiveness of this new approach.

2 Multiple Variable Splitting Reformulation of Linear SVMs

The approach introduced in this work consists of partitioning the dataset of p points into k subsets of, respectively, p_i, $i = 1, \ldots, k$, points (where $\sum_{i=1}^{k} p_i = p$). The points in each subset and their labels are assumed to be stored, respectively, row-wise in matrices $A^i \in \mathbb{R}^{p_i \times d}$ and diagonally in matrices $Y^i \in \mathbb{R}^{p_i \times p_i}$, $i = 1, \ldots, k$. Considering k smaller SVMs, each with its own variables (w^i, γ^i, s^i), $i = 1, \ldots, k$ (where $w^i \in \mathbb{R}^d$, $\gamma^i \in \mathbb{R}$ and $s^i \in \mathbb{R}^{p_i}$), problem (1) is equivalent to the following multiple variable splitting formulation with linking constraints:

$$\min_{(w^i, \gamma^i, s^i)} \quad \frac{1}{2} \left(\sum_{i=1}^{k} {w^i}^\top w^i \right) / k + \nu \sum_{i=1}^{k} {e^i}^\top s^i \tag{8a}$$

$$\text{subject to} \quad Y^i (A^i w^i + \gamma^i e^i) + s^i \geq e^i \qquad i = 1, \ldots, k \tag{8b}$$

$$s^i \geq 0 \qquad i = 1, \ldots, k \tag{8c}$$

$$w^i = w^{i+1}, \quad \gamma^i = \gamma^{i+1} \qquad i = 1, \ldots, k-1, \tag{8d}$$

where $e^i \in \mathbb{R}^{p_i}$ is a vector of ones. Linking constraints (8d) impose the same hyperplanes for the k SVMs. Slacks s^i represent the potential misclassification errors, so they are particular to the points of each subset and do not have to be included in the linking constraints.

Similarly, the one-class SVM problem (5) can be reformulated as

$$\min_{(w^i,\gamma^i,s^i)} \quad \frac{1}{2}\left(\sum_{i=1}^{k} {w^i}^\top w^i - \gamma^i\right)/k + \frac{1}{\nu p}\sum_{i=1}^{k} {e^i}^\top s^i \tag{9a}$$

$$\text{subject to} \quad A^i w^i - \gamma^i e^i + s^i \geq 0 \qquad i = 1, \ldots, k \tag{9b}$$

$$s^i \geq 0 \qquad i = 1, \ldots, k \tag{9c}$$

$$w^i = w^{i+1}, \quad \gamma^i = \gamma^{i+1} \qquad i = 1, \ldots, k-1. \tag{9d}$$

The constraints of problems (8) and (9) exhibit a primal block-angular structure. Putting aside the linking constraints (8d) and (9d), the solution of either (8) or (9) with an IPM requires k Cholesky factorizations involving $A^i \Theta^i {A^i}^\top \in \mathbb{R}^{p_i \times p_i}$, $i = 1, \ldots, k$, at each IPM iteration (Θ^i being a diagonal scaling matrix that depends on the particular iteration). If each subset has the same number of points, that is $p_i = p/k$, the complexity of the k Cholesky factorizations is

$$O\left(k\left(\frac{p}{k}\right)^3\right) = O\left(\frac{p^3}{k^2}\right) \ll O\left(p^3\right), \tag{10}$$

where $O\left(p^3\right)$ is the complexity of the Cholesky factorizations of the original formulations (1) and (5). Of course, to benefit from (10), we need an IPM that can efficiently deal with the linking constraints of primal block-angular optimization problems. Such an approach is summarized in the next section.

3 The IPM for the Multiple Variable Splitting Reformulation of SVMs

After transforming (8b) and (9b) into equality constraints by adding extra nonnegative variables $\xi^i \in \mathbb{R}^{p_i}$,

$$\begin{aligned} Y^i(A^i w^i + \gamma^i e^i) + s^i - \xi^i = e^i, \quad \xi^i \geq 0, \qquad & i = 1, \ldots, k \\ A^i w^i - \gamma^i e^i + s^i - \xi^i = 0, \quad \xi^i \geq 0, \qquad & i = 1, \ldots, k, \end{aligned}$$

problems (8) and (9) match the following general formulation of primal block-angular optimization problems:

$$\min \quad \sum_{i=0}^{k} f_i(x^i) \tag{11a}$$

$$\text{subject to} \quad \begin{bmatrix} M_1 & & & & \\ & M_2 & & & \\ & & \ddots & & \\ & & & M_k & \\ L_1 & L_2 & \dots & L_k & I \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^k \\ x^0 \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^k \\ b^0 \end{bmatrix} \tag{11b}$$

$$0 \le x_j^i \le u_j^i \quad j \notin \mathcal{F}^i, \quad x_j^i \text{ free } \quad j \in \mathcal{F}^i, \quad i = 0, \dots, k. \tag{11c}$$

Vectors $x^i = ({w^i}^\top \ {\gamma^i}^\top \ {s^i}^\top \ {\xi^i}^\top)^\top \in \mathbb{R}^{n_i = d+1+2p_i}$, $i = 1, \dots, k$, contain all the variables for the i-th SVM; and $\mathcal{C}^2 \ni f_i : \mathbb{R}^{n_i} \to \mathbb{R}$, $i = 0, \dots, k$, are convex separable functions. For SVM problems, they are quadratic functions:

$$f_i(x^i) = {c^i}^\top x^i + \frac{1}{2} {x^i}^\top Q_i x^i, \quad Q_i \succeq 0 \text{ and diagonal}, \qquad i = 1, \dots, k, \tag{12}$$

whereas for $i = 0$ we have $f_0(x^0) = 0$. Matrices $M_i \in \mathbb{R}^{m_i \times n_i}$ and $L_i \in \mathbb{R}^{l \times n_i}$, $i = 1, \dots, k$, respectively, define the block-diagonal and linking constraints, where $m_i = p_i$ (the number of points in the i-th SVM) and $l = (d+1)(k-1)$ is the number of linking constraints defined in either (8d) or (9d). Vector $b^i \in \mathbb{R}^{m_i}$, $i = 1, \dots, k$, is the right-hand side for each block of constraints, whereas $b^0 \in \mathbb{R}^l$ is for the linking constraints. In our case $b^i = e^i$ for two-class SVM and $b^i = 0$ for one-class SVM, $i = 1, \dots, k$, whereas $b^0 = 0$ in both problems. $x^0 \in \mathbb{R}^l$ are the slacks of the linking constraints. The sets $\mathcal{F}^i$ contain the indices of the free variables for each block (corresponding to w^i and γ^i). The upper bounds for each group of variables are $u^i \in \mathbb{R}^{n_i}$, $i = 0, \dots, k$; these upper bounds apply only to the components of x^i that are not in $\mathcal{F}^i$ (that is, they apply only to s^i and ξ^i), and in our problem $u^i = +\infty$ for all $i = 1, \dots, k$. For the linking constraints, we have $\mathcal{F}^0 = \emptyset$, that is, slacks x^0 are bounded—otherwise the linking constraints could be removed. For problems with equality linking constraints, as in our case, u^0 can be set to a very small (close to 0) value.

The total numbers of constraints and variables in (11) are thus, respectively, $m = l + \sum_{i=1}^k m_i$ and $n = l + \sum_{i=1}^k n_i$. Formulation (11) is a very general model which accommodates to many block-angular problems. In this work, problem (11) is solved by the specialized infeasible long-step primal-dual path-following IPM, which was initially introduced in [3] for multicommodity network flows and later extended to general primal block-angular problems [4, 6]. For the solution of SVM problems, we extended the implementation of this algorithm in order to deal with free variables, as described in [5].

For completeness, we will outline the path-following IPM used in this work, in order to derive the particular structure of the systems of equations to be solved. A detailed description of primal-dual path-following IPMs can be found in the monograph [22]. Problem (11) can be recast in general form as

$$\begin{array}{rl} \min & c^\top x + \dfrac{1}{2} x^\top Q x \\ \text{subject to} & Mx = b \\ & 0 \le x_i \le u_i \; i \notin \mathcal{F}, \quad x_i \text{ free } \; i \in \mathcal{F}, \end{array} \tag{13}$$

where $c \in \mathbb{R}^n$, $Q \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^m$, $M \in \mathbb{R}^{m \times n}$, and $\mathcal{F}$ denote the set of indices of free variables. Matrix Q is diagonal with nonnegative entries for SVM problems. Let $\zeta \in \mathbb{R}^m$, $\upsilon \in \mathbb{R}^{n-|\mathcal{F}|}$ and $\omega \in \mathbb{R}^{n-|\mathcal{F}|}$ be the vectors of Lagrange multipliers of, respectively, equality constraints, lower bounds, and upper bounds. To simplify the notation, given any vector $z \in \mathbb{R}^{n-|\mathcal{F}|}$, the vector $\tilde{z} \in \mathbb{R}^n$ will be defined as

$$\tilde{z}_i = \begin{cases} z_i & \text{for } i \notin \mathcal{F} \\ 0 & \text{for } i \in \mathcal{F} \end{cases};$$

and given any vector $z \in \mathbb{R}^n$, the matrix $Z \in \mathbb{R}^{n \times n}$ will be $Z = \text{diag}(z)$.

For any $\mu \in \mathbb{R}_+$, the central path can be derived as a solution of the μ-perturbed Karush–Kuhn–Tucker optimality conditions of (13):

$$r_b \equiv b - Mx = 0, \tag{14}$$

$$r_c \equiv Qx + c - M^\top \zeta - \tilde{\upsilon} + \tilde{\omega} = 0, \tag{15}$$

$$r_{x\upsilon} \equiv \mu \tilde{e} - X \tilde{\Upsilon} \tilde{e} = 0, \tag{16}$$

$$r_{x\omega} \equiv \mu \tilde{e} - (\tilde{U} - X) \tilde{\Omega} \tilde{e} = 0, \tag{17}$$

$$(x, \upsilon, \omega) \ge 0, \tag{18}$$

The primal-dual path-following method consists in solving the nonlinear system (14)–(18) by a sequence of damped Newton's directions (with step-length reduction to preserve the nonnegativity of variables), decreasing the value of μ at each iteration. On the left-hand side of (14)–(18), we have explicitly defined the residuals of the current iterate r_b, r_c, $r_{x\upsilon}$ and $r_{x\omega}$.

By performing a linear approximation of (14)–(18) around the current point, we obtain the Newton system in variables Δx, $\Delta \zeta$, $\Delta \upsilon$ and $\Delta \omega$. Applying Gaussian elimination the Newton system can be reduced to the *normal equations* form (see, for instance, [22] for details)

$$M \Theta M^\top \Delta \zeta = g, \tag{19}$$

where

$$\begin{aligned} g &= r_b + M\Theta(r_c + (\tilde{U} - X)^{-1} r_{x\omega} - X^{-1} r_{x\upsilon}) \in \mathbb{R}^m, \\ \Theta &= (Q + \tilde{\Omega}(\tilde{U} - X)^{-1} + \tilde{\Upsilon} X^{-1})^{-1} \in \mathbb{R}^n, \end{aligned} \tag{20}$$

The values of Δx, $\Delta \upsilon$ and $\Delta \omega$ can be easily computed once $\Delta \zeta$ is known. Note that Θ is a diagonal matrix since Q is diagonal.

Free variables in $\mathcal{F}$ do not have associated Lagrange multipliers in ω and υ, and then, according to (20), $\Theta_{\mathcal{F}} = Q_{\mathcal{F}}^{-1}$ (where $\Theta_{\mathcal{F}}$ and $Q_{\mathcal{F}}$ are the submatrices of Θ and Q associated with free variables). For the variables w^i that define the normal vector of the SVM hyperplane, this is not an issue, since those variables have a nonzero entry in matrix Q of the quadratic costs. However, the intercept γ^i of the SVM hyperplane has neither a multiplier in ω and υ nor a quadratic entry in Q; thus, its associated entry in the scaling matrix Θ is 0, making it singular. This can be fixed by using the regularization strategy for free variables described in [17]. A derivation and additional details about the normal equations can be found in [22].

Exploiting the block structure of M and Θ we have:

$$M\Theta M^\top \Delta\zeta = \left[\begin{array}{ccc|c} M_1\Theta_1 M_1^\top & & & M_1\Theta_1 L_1^\top \\ & \ddots & & \vdots \\ & & M_k\Theta_k M_k^\top & M_k\Theta_k L_k^\top \\ \hline L_1\Theta_1 M_1^\top & \dots & L_k\Theta_k M_k^\top & \Theta_0 + \sum_{i=1}^k L_i\Theta_i L_i^\top \end{array}\right] \Delta\zeta = \left[\begin{array}{cc} B & C \\ C^\top & D \end{array}\right]\left[\begin{array}{c} \Delta\zeta_1 \\ \Delta\zeta_2 \end{array}\right] = \left[\begin{array}{c} g_1 \\ g_2 \end{array}\right], \tag{21}$$

where $\Delta\zeta_1 \in \mathbb{R}^{\sum_{i=1}^k m_i}$ and $\Delta\zeta_2 \in \mathbb{R}^l$ are the components of $\Delta\zeta$ associated with, respectively, block and linking constraints; $\Theta_i = (Q_i + \tilde{\Omega}_i(\tilde{U}_i - X_i)^{-1} + \tilde{\Upsilon}_i X_i^{-1})^{-1}$, $i = 0, \dots, k$, are the blocks of Θ; and $g = (g_1^\top g_2^\top)^\top$ is the corresponding partition of the right-hand side g. We note that matrix B is comprised of k diagonal blocks $M_i\Theta_i M_i^\top$ for $i = 1, \dots, k$, each of them associated with one of the subsets in which the dataset of points was partitioned. By eliminating $\Delta\zeta_1$ from the first group of equations of (21), we obtain

$$(D - C^\top B^{-1} C)\Delta\zeta_2 = (g_2 - C^\top B^{-1} g_1) \tag{22a}$$

$$B\Delta\zeta_1 = (g_1 - C\Delta y_2). \tag{22b}$$

The specialized IPM for this class of problems solves (22) by performing k Cholesky factorizations for the k diagonal blocks of B and by using a preconditioned conjugate gradient (PCG) for (22a). System (22a) can be solved by PCG because matrix $D - C^\top B^{-1} C \in \mathbb{R}^{l\times l}$ of (22a) (whose dimension is the number of linking constraints) is symmetric and positive definite, since it is the Schur complement of the normal equations (21), which are symmetric and positive definite. A good preconditioner is, however, instrumental. We use the one introduced in [3], which is based on the *P-regular splitting theorem* [18]. $D - C^\top B^{-1} C$ is a *P-regular splitting*, i.e. it is symmetric and positive definite; D is nonsingular; and $D + C^\top B^{-1} C$ is positive definite. Therefore, the P-regular splitting theorem guarantees that

$$0 < \rho(D^{-1}(C^\top B^{-1} C)) < 1, \tag{23}$$

where $\rho(\cdot)$ denotes the spectral radius of a matrix (i.e. the maximum absolute eigenvalue). This allows us to compute the inverse of $D - C^\top B^{-1} C$ as the following infinite power series (see [3, Prop. 4] for a proof).

$$(D - C^\top B^{-1} C)^{-1} = \left(\sum_{i=0}^{\infty} (D^{-1}(C^\top B^{-1} C))^i \right) D^{-1}. \tag{24}$$

The preconditioner is thus obtained by truncating the infinite power series (24) at some term. In theory, the more terms that are considered, the fewer PCG iterations that are required, although at the expense of increasing the cost of each PCG iteration. Including only the first, and only the first and second terms of (24), the resulting preconditioners are, respectively, D^{-1} and $(I + D^{-1}(C^\top B^{-1} C))D^{-1}$. As observed in [5], D^{-1} generally provided the best results for most applications, and it will be our choice for solving SVM problems.

Although its performance is problem dependent, the effectiveness of the preconditioner obtained by truncating the infinite power series (24) most often depends on two criteria:

- First, the quality of the preconditioner relies on the spectral radius $\rho(D^{-1}(C^\top B^{-1} C))$, which is always in $(0, 1)$: the farther from 1, the better the preconditioner [6]. The value of the spectral radius strongly depends on the particular problem (even instance) being solved. Therefore, it is difficult to know a priori if the approach will be efficient for some particular application. However, there are a few results that justify its application for solving SVMs: Theorem 1 and Proposition 2 of [6] state that the preconditioner is more efficient for quadratic problems (such as SVMs) than for purely linear optimization problems.
- Secondly, the structure of matrix $D = \Theta_0 + \sum_{i=1}^{k} L_i \Theta_i L_i^\top$, since systems with this matrix have to be solved at each PCG iteration. Therefore, D has to be easily formed and factorized. We show in the next subsection that building and factorizing the matrices D that arise in SVMs are computationally fast operations.

3.1 The Structure of the Preconditioner *D*

According to (21), the preconditioner D is defined as

$$\mathbb{R}^{l \times l} \ni D = \Theta_0 + \sum_{i=1}^{k} L_i \Theta_i L_i^\top \tag{25}$$

where, from (8d) and (9d), the structure of $\begin{bmatrix} L_1 & \dots & L_k \end{bmatrix}$ is

$$\begin{bmatrix} L_1 & \dots & L_k \end{bmatrix} = \begin{array}{c} \begin{matrix} w^1, \gamma^1 & w^2, \gamma^2 & w^3, \gamma^3 & \cdots & & w^{k-1}, \gamma^{k-1} & w^k, \gamma^k \end{matrix} \\ \begin{bmatrix} I & -I & & & & & \\ & I & -I & & & & \\ & & & \ddots & & & \\ & & & & & I & -I \end{bmatrix} \end{array}. \tag{26}$$

Algorithm *Factorization of* $\sum_{i=1}^{k} L_i \Theta_i L_i^\top$ *(matrix* (27)*)*
Input: $\Theta_i, i = 1, \ldots k$
Output: Matrix $R : \sum_{i=1}^{k} L_i \Theta_i L_i^\top = RR^\top$, where

$$R = \begin{bmatrix} D_1 & & & & \\ S_2 & D_2 & & & \\ \ddots & \ddots & & & \\ & S_{k-2} & D_{k-2} & & \\ & & S_{k-1} & D_{k-1} \end{bmatrix}, \; S_i \text{ and } D_i \succ 0 \text{ diagonal}$$

$D_1 = (\Theta_1 + \Theta_2)^{1/2}$
for $i = 2$ **to** $k - 1$
$S_i = -\Theta_i D_1^{-1}$
$D_i = \left((\Theta_i + \Theta_{i+1}) - S_i^2\right)^{1/2}$
end_for
End_algorithm

Fig. 1 Algorithm for efficiently computing the factorization of matrix (27)

From (26) by block multiplication we get

$$\sum_{i=1}^{k} L_i \Theta_i L_i^\top = \begin{bmatrix} \Theta_1 + \Theta_2 & -\Theta_2 & & & \\ -\Theta_2 & \Theta_2 + \Theta_3 & -\Theta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & -\Theta_{k-2} & \Theta_{k-2} + \Theta_{k-1} & -\Theta_{k-1} \\ & & & -\Theta_{k-1} & \Theta_{k-1} + \Theta_k \end{bmatrix}. \quad (27)$$

Therefore, from (25), the preconditioner D is a *t-shifted (symmetric and positive definite) tridiagonal matrix*, where t is the number of split variables (in the SVM problem, $t = d + 1$ is the number of components in (w, γ) defining the separation hyperplane). A t-shifted tridiagonal matrix is a generalization of a tridiagonal matrix where the superdiagonal (nonzero diagonal above the main diagonal) and subdiagonal (nonzero diagonal below the main diagonal) are shifted t positions from the main diagonal, i.e. elements (i, j) are non-zero only if $|i - j|$ is either 0 or t. Matrices with such a structure can be efficiently factorized with zero fill-in by extending a standard factorization for tridiagonal matrices. The algorithm in Fig. 1 shows the efficient factorization of matrix (27).

The above discussion is summarized in the following result:

Proposition 1 *For any two-class or one-class SVM problem based, respectively, on the splitting formulations* (8) *and* (9)*, the preconditioner D defined in* (25) *is a* $(d+1)$*-shifted tridiagonal matrix of dimension* $(d + 1)(k - 1)$*, where* $d + 1$ *is the number of components in* (w, γ) *and k is the number of subsets in which the points of the SVM were partitioned.*

Proof From (25), D is the sum of the diagonal matrix Θ_0 and $\sum_{i=1}^{k} L_i \Theta_i L_i^\top$. Using the structure of $[L_1 \ldots L_k]$ in (25), by block multiplication we get that $\sum_{i=1}^{k} L_i \Theta_i L_i^\top$ is the tridiagonal matrix (27), where each diagonal matrix Θ_i, $i = 1, \ldots, k$ has dimension $d + 1$. Therefore, D is a $(d + 1)$-shifted tridiagonal matrix. □

4 Computational Results

The specialized algorithm for SVMs detailed in Sect. 3 has been coded in C++ using the BlockIP package [5], which is an implementation of the IPM for block-angular problems. The resulting code will be referred to as SVM-BlockIP. SVM-BlockIP solves both the two-class and one-class SVM models (8) and (9). The executable file of SVM-BlockIP can be downloaded at http://www-eio.upc.edu/~jcastro/SVM-BlockIP.html.

SVM-BlockIP is compared with the following solvers:

- The standard primal-dual barrier algorithm in CPLEX 20.1. In general this interior-point variant is faster than the homogeneous-self-dual one, especially when a loose optimality tolerance is considered (which is our case, as discussed below). For a fair comparison, both the original compact SVM models (1) and (5) as well as the new splitting ones (8) and (9) will be solved with CPLEX 20.1.
- SVM-OOPS [21] is a very efficient IPM based on a separable reformulation of the dual of the two-class SVM compact model (4). SVM-OOPS does not solve one-class SVM problems.
- LIBSVM [7] solves the dual compact models (4) and (6), so it can be used for both two-class and one-class SVMs. It is based on a specialized algorithm developed in the machine learning community for SVMs, which is called sequential minimal optimization [19].
- LIBLINEAR [11] solves the compact models of two-class and one-class SVMs. For two-class SVMs, it can solve either the dual or the primal model, but without the γ variable (so the models solved by LIBLINEAR are a bit different—and simpler—than those considered by the other solvers). For one-class SVMs, it solves the dual (6) with a coordinate descent algorithm.

The same parameters (e.g. optimality tolerance) were used for all the solvers.

It is in general not desirable (and indeed, not recommended) to compute an optimal solution to an SVM optimization problem using a tight optimality tolerance because, otherwise, the plane (w^*, γ^*) may excessively fit to the dataset of points a_i, $i = 1, \ldots, p$ (named the *training dataset*), and it might not be able to properly classify a new and different set of points (named the *testing dataset*). This phenomenon is named *overfitting* in the data science community [10]. For this reason, SVM-BlockIP and the rest of solvers will be executed with an optimality tolerance of 10^{-1}. It is worth noting that loose optimality tolerances are more advantageous for SVM-BlockIP than for the other interior-point solvers that rely only on Cholesky factorizations (CPLEX 20.1 and SVM-OOPS), namely because it has been observed [3–6] that SVM-BlockIP needs a greater number of PCG iterations for computing the Newton direction when approaching the optimal solution. A loose tolerance thus avoids SVM-BlockIP's last and most expensive interior-point iterations.

A loose optimality tolerance also allows using loose tolerances for solving PCG systems. Indeed, the requested PCG tolerance is one of the parameters that most influence the efficiency of the specialized IPM. The PCG tolerance in the BlockIP solver is dynamically updated at each interior-point iteration i as $\epsilon_i = \max\{\beta\epsilon_{i-1}, \min_\epsilon\}$, where ϵ_0 is the initial tolerance, $\min_\epsilon$ is the minimum allowed tolerance, and $\beta \in [0, 1]$

is a tolerance reduction factor at each interior-point iteration. For SVM-BlockIP, we used $\epsilon_0 = 10^{-2}$ and $\beta = 1$, that is, a tolerance of 10^{-2} was used for all the interior-point iterations. It is known that the resulting inexact Newton direction does not seriously affect the convergence properties of IPMs [16].

For the computational results, we considered a set of 19 standard SVM instances. Their dimensions are reported in Table 1. Columns p and d show the number of points and features, respectively. The instances are divided into two groups according to their number of features, since previous interior-point approaches for SVM could handle only instances with a few features. Column k is the number of subsets of points considered, and each instance was tested with different values. Rows with $k = 1$ (that is, without splitting) refer to the compact formulations (1) and (5), while for $k > 1$ the row is associated with the splitting models (8) and (9). The cases with $k = 1$ (compact model) were solved with CPLEX, SVM-OOPS, LIBSVM and LIBLINEAR; when $k > 1$, only CPLEX and SVM-BlockIP can be used. Since very dense matrices $A^i A^{i\top}$ in the interior-point method may be provided by submatrices A^i (which are related to the splitting model's i-th subset of p_i points), the value of $k > 1$ was selected so that p/k ($\approx p_i$) was always less than 1000 (we observed that the dense factorization of $A^i A^{i\top}$ was too expensive for dimensions greater than 1000). For instances with a large number of points, two values of k were tested (one being ten times greater than the other); increasing k reduces the dimensions of systems $A^i A^{i\top}$ at the expense of increasing the number of variables and linking constraints of the splitting formulation. As a rule of thumb, as as will be shown below, the best results with the splitting formulation are obtained with the greatest k when d is small, and with the smallest k when d is large. Finally columns "n.vars.", "n.cons." and l give, respectively, the numbers of variables, constraints (excluding linking constraints) and linking constraints, which are computed, also respectively, as $(d + 1)k + 2p$, p, and $(d + 1)(k - 1)$. This set of SVM instances includes the full version of the four largest (out of the five) cases tested in [21]. It is worth noting that SVM-BlockIP was extended with dense matrix operations (in addition to the default sparse ones in the BlockIP IPM package) in order to handle problems with very dense matrices of points A^i. Executions with both sparse and dense matrices were considered only for four of the instances in Table 1 (namely, "gisette", "madelon", "sensit", and "usps").

The instances tested are in the format used by the standard SVM packages LIBSVM [7] and LIBLINEAR [11], and they were retrieved from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. Points $a_i \in \mathbb{R}^d$, $i = 1, \ldots, p$ in those instances were properly scaled by the authors of LIBSVM and LIBLINEAR, such that most features are in the range of $[-1, 1]$. In those instances, for two-class SVM, the value of parameter ν was 1 for all the solvers; for one-class SVM, $\nu = 0.1$ was used.

From the previous original instances, we generated a second set of cases by applying an alternative (linear) scaling to the original features. In most cases of the new linear scaling, features were concentrated within the interval $[0, 0.001]$. This second set of instances has the same dimensions as those in Table 1. As a result of the new scaling, the optimal normal vector w^* will take larger values, so the quadratic term in the objective function will be larger. To compensate for this fact, a value of $\nu = 1000$ was used for two-class SVM. For one-class SVM, we used the same value of $\nu = 0.1$ that

Table 1 Dimensions of SVM instances

	Instance	p	d	k	n.vars.	n.cons.	l
small d (few features)	a9a	32,561	123	1	65,246	32,561	0
				100	77,522	32,561	12,276
				1000	189,122	32,561	123,876
	australian	690	14	1	1395	690	0
				2	1410	690	15
	covtype	581,012	54	1	1,162,079	581,012	0
				1000	1,217,024	581,012	54,945
				10,000	1,712,024	581,012	549,945
	ijcnn1	49,990	22	1	100,003	49,990	0
				100	102,280	49,990	2277
				1000	122,980	49,990	22,977
	madelon	2000	500	1	4501	2000	0
				10	9010	2000	4509
	mnist-ge5-lt5	60,000	780	1	120,781	60,000	0
				200	276,200	60,000	155,419
				2000	1,682,000	60,000	1,561,219
	mnist-odd-even	60,000	780	1	120,781	60,000	0
				200	276,200	60,000	155,419
				2000	1,682,000	60,000	1,561,219
	mushrooms	8124	112	1	16,361	8124	0
				20	18,508	8124	2147
	sensit	78,823	100	1	157,747	78,823	0
				100	167,746	78,823	9999
				1000	258,646	78,823	100,899

Table 1 continued

	Instance	p	d	k	n.vars.	n.cons.	l
	usps	7291	256	1	14,839	7291	0
				10	17,152	7291	2313
				100	40,282	7291	25,443
	w1a	2477	300	1	5255	2477	0
				10	7964	2477	2709
	w4a	7366	300	1	15,033	7366	0
				30	23,762	7366	8729
	w8a	49,749	300	1	99,799	49,749	0
				200	159,698	49,749	59,899
large d	colon-cancer	62	2000	1	2125	62	0
				10	20,134	62	18,009
	gisette	6000	5000	1	17,001	6000	0
				10	62,010	6000	45,009
				100	512,100	6000	495,099
	leu	38	7129	1	7206	38	0
				2	14,336	38	7130
	news20	19,996	1,355,191	1	1,395,184	19,996	0
				40	54,247,672	19,996	52,852,488
	rcv1	20,242	47,236	1	87,721	20,242	0
				40	1,929,964	20,242	1,842,243
				400	18,935,284	20,242	18,847,563
	real-sim	72,309	20,958	1	165,577	72,309	0
				100	2,240,518	72,309	2,074,941
				1000	21,103,618	72,309	20,938,041

was used for the original instances, since ν is related to the upper bound on the fraction of detected outliers in the one-class SVM problem. The increase in the quadratic term due to the new scaling turned out to be very advantageous for SVM-BlockIP, since (as was proven in [6, Prop. 2]) the quadratic terms in the objective function of (11) reduce the spectral radius (23), thus making the preconditioner more efficient. The set of scaled instances can be retrieved from the SVM-BlockIP webpage http://www-eio.upc.edu/~jcastro/SVM-BlockIP.html.

The next two subsections show the computational results for, respectively, two-class SVM and one-class SVM, using both the original datasets, and those with the new scaling. All the computational experiments in this work were carried out on a DELL PowerEdge R7525 server with two 2.4 GHz AMD EPYC 7532 CPUs (128 total cores) and 768 Gigabytes of RAM, running on a GNU/Linux operating system (openSuse 15.3), without exploitation of multithreading capabilities.

4.1 Results for Two-Class SVM Instances

Tables 2 and 3 show the results obtained for two-class SVM with, respectively, the original and scaled instances. For all the solvers (namely, SVM-BlockIP, CPLEX 20.1, LIBSVM, and LIBLINEAR), the tables provide: the number of iterations (columns "it"), solution time (columns "CPU"), objective function achieved (columns "obj"), and accuracy of the solution provided (columns "acc%"). The accuracy is the percentage of correctly classified points (of the testing dataset) using the hyperplane provided by the optimal solution (which was computed with the training dataset); that is, the accuracy is related to the the optimal solution's usefulness for classification purposes. For SVM-BlockIP, the tables also provide the overall number of PCG iterations (columns "PCGit"). The CPU time of the fastest execution for each instance is marked in boldface, excluding the LIBLINEAR time, since it solved the simpler problem (7) whereas the other solvers dealt with (1) or (4).

SVM-BlockIP allows computing both the Newton direction (19) and the predictor-corrector direction [22], both of which were tried for solving SVM problems. Although, in general, predictor-corrector directions are not competitive for PCG-based IPMs because they force using twice the PCG at each interior-point iteration, in some cases they provided the fastest solution. Those cases are marked with an "*" in their "CPU" columns.

From Table 2, it is observed that SVM-BlockIP was not competitive against the other solvers for instances with a small number of features (first rows in the table). In general, SVM-OOPS provided the fastest executions in six of these instances, CPLEX in four, and LIBSVM in three. All solvers converged to solutions of similar objective functions for all the instances but three (namely "covtype", "madelon", and "mushrooms"), in which LIBLINEAR and LIBSVM (after a large number of iterations) stopped at non-optimal points. However, it is worth noting that the accuracy of LIBLINEAR and LIBSVM in two of these three instances was still good. Furthermore, when the number of features increase (last rows of Table 2), SVM-OOPS was unable to solve five out of the six instances; LIBSVM was the fastest approach in three of these instances; and CPLEX in two (but in those two, SVM-BlockIP reported a similar time). In two cases

Table 2 Results for two-class SVM original instances (in boldface the CPU of fastest execution for each instance)

Instance	k	SVM-BlockIP					CPLEX 20.1			SVM-OOPS				LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%	it	CPU	obj	acc%
a9a	1	—					18	**0.6**	11,433.4	7	1.0	12,047.3	85.0	31,620	45.2	11,430.8	85.0	349	0.1	11,432.7	85.0
	100	28	649	11.2	11,793.0	85.0	10	17.6	11,436.7	—				—				—			
	1000	35	3213	11.6	11,080.6	85.0	10	23.2	11,761.9	—				—				—			
australian	1	—					10	0.1	200.0	6	**0.0**	213.4	90.3	397	**0.0**	199.0	86.0	104	0.0	199.6	86.0
	2	15	77	0.1	215.2	87.1	10	0.1	200.3	—				—				—			
covtype	1	—					‡			6	**8.3**	0.0	100.0	262,600	13,566.0	337,918.5§	76.3	237	3.3	337,939.7§	76.3
	1000	19	240	215.5	0.0	51.2	6	1889.3	0.0	—				—				—			
	10,000	32	765	73.2*	0.0	51.2	7	57.2	0.0	—				—				—			
ijcnn1	1	—					18	**0.6**	8590.2	11	0.7	9100.5	92.2	9384	16.5	8587.8	92.1	88	0.1	8589.9	92.1
	100	31	2073	58.2*	8829.1	92.1	11	20.9	8990.2	—				—				—			
	1000	67	57,450	261.6	8942.2	92.2	10	16.7	8929.5	—				—				—			
madelon	1	—					10	8.3	1144.7	5	**0.8**	1174.1	53.8	10,000,000	39.6	19,209.3§	52.0	100,000	134.4	1.2§	55.3
	10	29	26,088	48.7	1102.4	57.0	10	7.4	1116.6	—				—				—			
mnist-ge5-lt5	1	—					12	**30.8**	19,279.5	10	53.0	20,220.7	88.1	500,291	14,259.5	19,274.4	88.0	2300	7.6	19,278.6	88.0
	200	38	1564	80.1	19,350.8	88.0	12	541.6	19,292.0	—				—				—			
	2000	200	278,228	8765.2	19,775.8	88.0	14	792.7	19,285.8	—				—				—			
mnist-odd-even	1	—					11	**28.8**	14,717.8	10	53.8	15,666.2	90.5	373,829	10,031.0	14,713.8	90.4	1914	5.6	14,717.1	90.4
	200	56	2572	120.8	14,619.8	90.4	11	514.7	15,255.7	—				—				—			
	2000	200	283,027	8725.3	15,009.9	90.4	15	811.7	14,713.9	—				—				—			
mushrooms	1	—					‡			4	**0.2**	0.0	100.0	188	0.1	7.6§	100.0	43	0.0	7.6§	100.0
	20	13	126	1.6	0.0	48.2	6	1.7	0.0	—				—				—			
sensit	1	—					18	8.9	27,120.0	7	**5.9**	28,334.4	85.2	44,685	1082.6	27,113.0	85.8	1921	7.1	27,117.3	85.8

Table 2 continued

Instance	k	SVM-BlockIP					CPLEX 20.1			SVM-OOPS				LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%	it	CPU	obj	acc%
	100	14	425	113.0*	27,837.0	85.8	10	67.5	27,303.1	—				—				—			
	1000	22	377	20.0	26,995.9	85.9	10	21.5	27,279.3	—				—				—			
usps	1	—					8	137.1	1412.6	9	**1.7**	1500.1	90.1	45,328	7.1	1412.1	89.7	2583	1.1	1413.8	90.0
	10	15	504	21.4*	1456.8	89.8	9	12.5	1412.7	—				—				—			
	100	28	1595	8.2	1409.1	89.5	12	11.9	1412.7	—				—				—			
w1a	1	—					9	1.5	62.9	10	0.4	67.8	97.8	806	**0.0**	62.9	97.7	53	0.0	63.5	97.7
	10	28	827	0.7	63.0	97.8	10	0.6	63.2	—				—				—			
w4a	1	—					9	13.1	211.7	12	1.2	229.0	98.4	2464	**0.2**	211.4	98.4	99	0.0	212.2	98.4
	30	48	2911	5.9	204.0	98.4	11	2.4	223.7	—				—				—			
w8a	1	—					15	1.6	1579.3	41	27.2	1602.9	98.7	16,604	**10.8**	1485.0	98.7	395	0.3	1487.3	98.7
	200	138	196,287	2251.7	1382.8	98.7	16	67.6	1566.8	—				—				—			
colon-cancer	1	—					9	**0.0**	0.0	4	2.6	**0.0**	100.0	67	0.0	**0.0**	100.0	17	0.0	0.0	100.0
	10	9	419	0.2	0.0	100.0	6	0.2	0.0	—				—				—			
gisette	1	—					7	478.0	0.7	†				2313	**41.3**	0.7	97.4	75	3.8	0.7	97.6
	10	12	677	252.2*	0.7	97.8	7	813.9	0.7	—				—				—			
	100	28	4532	257.3	0.7	97.7	8	7511.4	0.7	—				—				—			
leu	1	—					10	**0.1**	0.0	†				34	**0.1**	0.0	82.4	9	0.0	0.0	79.4
	2	18	84	0.2*	0.0	85.3	10	0.2	0.0	—				—				—			
news20	1	—					7	1926.3	2561.3	†				6636	**468.1**	2558.1	99.6	25	1.6	2561.1	99.6

Table 2 continued

Instance	k	SVM-BlockIP					CPLEX 20.1			SVM-OOPS				LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%	it	CPU	obj	acc%
	40	104	1155	1305.9	2714.3	99.6	8	3010.4	2561.3	—				—				—			
rcv1	1	—					9	2475.4	1745.8	†				4023	78.0	1743.6	96.3	27	0.3	1745.4	96.3
	40	52	117	**23.5**	1748.1	96.3	7	2086.1	1746.7	—				—				—			
	400	111	6341	1638.2	1802.1	96.3	8	3046.0	1745.7	—				—				—			
real-sim	1	—					8	91,482.1	5345.1	†				10,873	659.6	5339.7	98.9	30	0.8	5344.7	98.9
	100	171	4699	**524.9**	2795.6	97.8	11	136,406.8	5345.3	—				—				—			
	1000	200	35,667	9651.0	4871.2	98.9	10	136,548.7	5346.8	—				—				—			

—Particular combination of (solver,k) is either not available (for SVM-OOPS, LIBSVM, LIBLINEAR) or it was not run (for SVM-BlockIP)

*Best result with SVM-BlockIP obtained with predictor-corrector direction

†Execution failed with SVM-OOPS

‡Wrong result: all columns and constraints eliminated by CPLEX 20.1 presolver

§LIBSVM or LIBLINEAR stopped at a non-optimal point

Table 3 Results for two-class SVM scaled instances (in boldface the CPU of fastest execution for each instance)

Instance	k	SVM-BlockIP					CPLEX 20.1			SVM-OOPS				LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%	it	CPU	obj	acc%
a9a	1	—					15	**0.5**	1.3e+07	13	1.7	1.4e+07	84.1	7943	32.7	1.3e+07	83.9	22,838	4.9	1.3e+07	84.0
	100	12	20	3.3	1.4e+07	84.1	12	22.1	1.3e+07	—				—				—			
	1000	13	30	0.7	1.3e+07	84.0	18	39.5	1.3e+07	—				—				—			
australian	1	—					5	0.1	5.8e+05	5	**0.0**	5.9e+05	57.0	307	**0.0**	5.8e+05	57.0	13,299	0.1	5.8e+05	57.0
	2	6	9	**0.0**	5.9e+05	57.0	9	0.1	5.8e+05	—				—				—			
covtype	1	—					10	5.2	3.8e+08	9	12.6	3.9e+08	70.5	223,942	9773.6	3.7e+08	68.8	41,435	483.1	3.7e+08	68.8
	1000	17	102	169.8	3.4e+08	54.1	86	23,422.4	4.0e+08	—				—				—			
	10,000	23	132	23.9	2.1e+08	52.0	189	1084.4	3.8e+08	—				—				—			
ijcnn1	1	—					10	0.4	1.0e+07	6	0.6	1.0e+07	90.5	5278	13.4	9.7e+06	90.5	12,860	25.5	9.7e+06	90.5
	100	7	27	7.3*	1.0e+07	90.5	21	40.0	1.0e+07	—				—				—			
	1000	6	32	1.1*	1.0e+07	90.5	10	15.7	9.7e+06	—				—				—			
madelon	1	—					4	3.8	2.0e+06	4	0.9	2.0e+06	59.7	1000	2.6	2.0e+06	58.5	3809	1.8	2.0e+06	56.8
	10	4	18	1.6*	1.9e+06	57.3	7	8.6	2.0e+06	—				—				—			
mnist-ge5-lt5	1	—					9	24.7	2.5e+07	14	74.4	2.6e+07	86.0	15,321	1064.2	2.5e+07	86.2	24,334	59.9	2.5e+07	86.2
	200	14	41	20.2	2.6e+07	86.2	11	511.9	2.5e+07	—				—				—			
	2000	38	130	15.6	2.4e+07	86.0	19	947.0	2.5e+07	—				—				—			
mnist-odd-even	1	—					11	28.8	1.9e+07	17	87.1	2.0e+07	89.1	11,931	817.4	1.9e+07	89.0	19,579	45.9	1.9e+07	89.0
	200	16	48	21.8	2.0e+07	89.0	11	515.6	1.9e+07	—				—				—			
	2000	28	118	12.5	1.8e+07	89.0	18	922.6	1.9e+07	—				—				—			
mushrooms	1	—					7	0.1	1.6e+06	10	0.3	1.6e+06	98.1	1190	1.2	1.6e+06	98.3	17,092	2.8	1.6e+06	98.2
	20	20	39	1.7	1.3e+06	96.3	16	2.9	1.6e+06	—				—				—			
sensit	1	—					21	9.8	3.0e+07	10	7.5	3.1e+07	84.6	19,172	853.2	3.0e+07	84.7	33,577	92.3	3.0e+07	84.7

Table 3 continued

Instance	k	SVM-BlockIP					CPLEX 20.1			SVM-OOPS				LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%	it	CPU	obj	acc%
	100	7	21	65.3*	3.1e+07	84.6	13	85.5	3.2e+07	—				—				—			
	1000	10	35	9.7*	3.0e+07	84.7	19	53.7	3.0e+07	—				—				—			
usps	1	—					2	44.2	3.3e+06	8	1.6	3.4e+06	86.4	2177	11.1	3.3e+06	86.6	21,500	9.2	3.3e+06	86.6
	10	7	28	10.4*	3.3e+06	86.4	10	23.7	3.3e+06	—				—				—			
	100	10	22	1.4	3.4e+06	86.6	11	18.7	3.3e+06	—				—				—			
w1a	1	—					14	2.3	1.4e+05	5	0.2	1.5e+05	97.0	74	0.0	1.4e+05	97.0	2121	0.2	1.4e+05	97.0
	10	9	22	0.1	1.6e+05	97.0	11	0.6	1.4e+05	—				—				—			
w4a	1	—					7	10.4	4.4e+05	7	0.8	4.6e+05	97.0	231	0.2	4.3e+05	97.0	10,674	2.3	4.3e+05	97.0
	30	9	82	0.5*	4.4e+05	97.0	20	3.6	4.3e+05	—				—				—			
w8a	1	—					17	1.8	2.9e+06	34	25.3	3.1e+06	97.0	1785	7.8	2.9e+06	97.0	17,200	15.8	2.9e+06	97.0
	200	10	119	4.3*	4.3e+05	97.0	28	110.1	2.9e+06	—				—				—			
colon-cancer	1	—					7	0.0	3.1e+01	11	6.2	3.3e+01	100.0	75	0.0	3.1e+01	100.0	62	0.0	3.1e+01	100.0
	10	10	577	0.2	3.1e+01	100.0	6	0.2	3.1e+01	—				—				—			
gisette	1	—					5	54.0	1.0e+06	11	314.9	1.0e+06	97.2	919	42.3	9.8e+05	97.4	10,243	14.1	9.8e+05	97.4
	10	9	32	8.1*	9.8e+05	97.5	7	149.4	9.8e+05	—				—				—			
	100	19	58	3.2	9.6e+05	97.3	8	486.0	9.8e+05	—				—				—			
leu	1	—					5	0.1	3.0e−02	†				31	0.1	1.0e−02	88.2	11	0.0	1.0e−02	88.2
	2	14	78	0.1	1.0e−02	91.2	6	0.1	1.0e−02	—				—				—			
news20	1	—					3	968.3	1.1e+07	†				8202	511.4	1.0e+07	94.4	12,967	111.8	1.0e+07	94.3

Table 3 continued

Instance	k	SVM-BlockIP					CPLEX 20.1			SVM-OOPS				LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%	it	CPU	obj	acc%
	40	6	54	84.8*	8.9e+05	95.7	19	6796.4	1.0e+07	—				—				—			
rcv1	1	—					4	1236.7	1.3e+07	†				8321	151.4	1.3e+07	91.9	18,527	12.2	1.3e+07	91.5
	40	12	34	7.9*	1.2e+07	91.8	11	3223.8	1.3e+07	—				—				—			
	400	13	29	29.5	1.1e+07	91.3	16	5817.2	1.3e+07	—				—				—			
real-sim	1	—					3	40,484.8	4.2e+07	†				24,469	1777.8	3.9e+07	72.4	27,407	91.2	3.9e+07	72.4
	100	4	32	14.4*	3.8e+05	69.2	27	320,107.0	3.9e+07	—				—				—			
	1000	32	287	299.6	2.0e+04	69.2	38	495,538.2	3.9e+07	—				—				—			

—Particular combination of (solver,k) is either not available (for SVM-OOPS, LIBSVM, LIBLINEAR) or it was not run (for SVM-BlockIP)
*Best result with SVM-BlockIP obtained with predictor-corrector direction
†Execution failed with SVM-OOPS

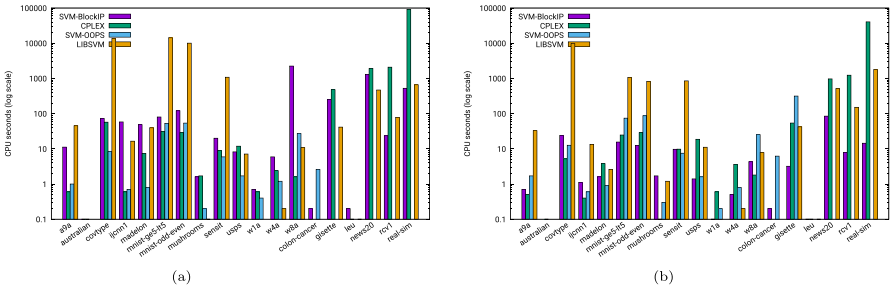

Fig. 2 CPU seconds (in log scale) required by SVM-BlockIP, CPLEX, SVM-OOPS and LIBSVM for two-class SVM, with the original instances in Table 2 (plot **a**) and the scaled instances in Table 3 (plot **b**)

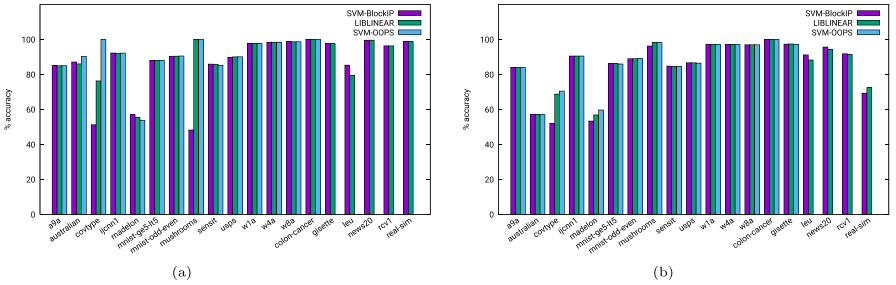

Fig. 3 Accuracies (in %) provided by solutions with SVM-BlockIP, LIBLINEAR and SVM-OOPS for two-class SVM, with the original instances in Table 2 (plot **a**) and the scaled instances in Table 3 (plot **b**)

(namely "rcv1" and "real-sim"), SVM-BlockIP obtained the fastest solutions in 23.5 and 524.9 seconds while CPLEX needed 2086.1 and 91,482.1 seconds. The fastest solution times—for any k—with SVM-BlockIP, CPLEX, SVM-OOPS and LIBSVM, are summarized in plot (a) of Fig. 2. LIBLINEAR was excluded from the comparison because it solved the slightly different problem (7). CPU times are in log scale. Note that there is no information for SVM-OOPS and the five rightmost instances, since, due to their large number of features, it could not solve them. Similarly, plot (a) of Fig. 3 shows the accuracy of the solutions obtained with SVM-BlockIP, LIBLINEAR (whose accuracies are similar to those of LIBSVM), and SVM-OOPS. It is evident that, in general, all methods provided similar accuracies (unlike for "covtype" and "mushrooms", where SVM-BlockIP underperformed the other solvers).

The results in Table 3 show a different behaviour of the solvers in the scaled dataset. For the instances with a few features, CPLEX, SVM-OOPS and LIBSVM were not significantly affected; for the instances with large d (last rows in the table), the CPU times increased notably for CPLEX and LIBSVM, whereas SVM-OOPS was once again unable to solve most of the problems. The coordinate gradient descent algorithm of LIBLINEAR significantly increased the CPU time for all the instances, independently of the number of features. However, the CPU times for SVM-BlockIP dropped drastically, thereby allowing it to solve the scaled dataset in a fraction of the times it required for the original dataset, which are reported in Table 2. SVM-BlockIP was the most efficient approach (including LIBLINEAR) in most cases, especially when the

number of features was large. For example, for the instances "rcv1" and "real-sim", SVM-BlockIP required, respectively, 7.9 and 14.4 seconds whereas CPLEX needed, respectively, 1236.7 and 40,484.8 seconds for $k = 1$, and 3223.8 and 320,107.0 seconds for the same $k > 1$ used with SVM-BlockIP. This fact can be explained by the higher importance of the quadratic term in the objective function due to the scaling (which is reflected in the larger objective values in Table 3 as compared to those in Table 2). Plot (b) of Fig. 2 shows the best CPU time, for any value of k, of SVM-BlockIP, CPLEX, SVM-OOPS and LIBSVM for the scaled dataset. LIBLINEAR is again excluded because it solves the simpler problem (7). Comparing plots (a) and (b) of Fig. 2 is clearly observed that the solution times with SVM-BlockIP significantly dropped for the scaled dataset. The downside of the scaling was that the accuracy decreased slightly in several instances (although it increased in a few, such as for problem "leu"), as can be observed in plot (b) of Fig. 3.

4.2 Results for One-Class SVM Instances

Tables 4 and 5 give the results of one-class SVM for, respectively, the original and scaled instances. SVM-OOPS does not solve the one-class SVM problem, so it is excluded from the comparison in those tables. The meaning of the columns is the same as in the previous Tables 2 and 3. For one-class SVM, the accuracy is measured as the percentage of dataset points that are not considered novelty or outliers; that is, 100 minus the accuracy is the percentage of detected outliers or novelty points. Since a value of $\nu = 0.1$ was used for one-class SVM (which is an upper bound on the fraction of detected outliers), accuracies should theoretically be greater than or equal to 90%.

Looking at Tables 4 and 5, it is clearly observed that LIBSVM and LIBLINEAR could not solve any instance, and their objective values were very different from those reported by CPLEX and SVM-BlockIP (which, in addition, were similar). Indeed, the solutions reported by LIBSVM and LIBLINEAR had very poor accuracy, usually around 50%, which means that the reported hyperplane is not useful for outlier or novelty detection. Such a different behaviour of LIBSVM and LIBLINEAR between two-class (where they provided high-quality hyperplanes) and one-class SVM is likely explained by the existence of constraint (6b), which complicates solving (6) by means of a coordinate gradient algorithm.

Unlike LIBSVM and LIBLINEAR, SVM-BlockIP was able to compute a fast and good solution for all the instances. For the original datasets in Table 4, SVM-BlockIP and CPLEX had similar performance for the instances with few features. However, for the instances with a large number of features (last rows in Table 4), SVM-BlockIP was generally much more efficient than CPLEX. For example, for the cases "gisette", "news20", "rcv1", and "real-sim" the best SVM-BlockIP times were, respectively, 40.5, 109.8, 7.4 and 22.2 seconds, whereas CPLEX required, respectively, 1104.9, 3141.7, 2624.1 and 71,227.9 seconds. This difference in performance between SVM-BlockIP and CPLEX slightly increased even for the scaled instances in Table 5. The fastest executions—for any value of k—with SVM-BlockIP and CPLEX are shown in plots (a) (for the original instances) and (b) (for the scaled instances) of Fig. 4. The

Table 4 Results for one-class SVM original instances (in boldface the CPU of fastest execution for each instance)

Instance	k	SVM-BlockIP					CPLEX 20.1			LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%
a9a	1	—					13	0.5	−2.0	3030	17.2	21,622,408.0§	28.3	29	0.0	21,622,408.0§	28.3
	100	12	153	5.0*	−2.0	91.2	7	13.7	−2.0	—				—			
	1000	25	776	3.4	−2.0	90.8	7	18.1	−2.0	—				—			
australian	1	—					11	0.1	−1.2	87	0.0	5703.2§	45.2	58	0.0	5703.2§	45.2
	2	18	150	0.1	−1.1	97.8	10	0.1	−1.2	—				—			
covtype	1	—					11	5.7	−1.0	55,701	3270.5	3,759,261,358.0§	45.4	29	1.0	3,759,261,358.1§	45.4
	1000	12	301	201.3*	−1.0	93.7	8	2460.0	−1.0	—				—			
	10,000	23	554	49.7	−1.0	89.9	9	68.6	−1.0	—				—			
ijcnn1	1	—					7	0.4	−0.0	8296	19.4	1,569,167.4§	14.0	357	0.1	1,569,167.4§	14.0
	100	7	319	13.1*	−0.0	95.5	6	13.3	−0.0	—				—			
	1000	13	1247	7.1	−0.0	99.9	6	10.7	−0.0	—				—			
madelon	1	—					12	9.8	−59,006,500.3	181	0.5	2,360,283,065,691.2§	52.2	4	0.1	2,360,283,065,650.2§	52.2
	10	18	101	3.0	−1,680,540.0	52.5	13	9.3	−3,857,346.3	—				—			
mnist-ge5-lt5	1	—					13	33.6	−5.5	5471	520.1	206,442,454.4§	51.5	39	1.1	206,442,454.4§	51.5
	200	28	436	41.0	−5.6	91.7	11	518.1	−5.7	—				—			
	2000	36	1887	71.5	−5.7	91.3	12	720.5	−5.7	—				—			
mnist-odd-even	1	—					13	32.4	−5.5	5471	519.3	206,442,454.4§	52.3	39	1.1	206,442,454.4§	52.3
	200	28	436	41.1	−5.6	91.7	11	518.4	−5.7	—				—			
	2000	36	1887	72.2	−5.7	91.3	12	721.9	−5.7	—				—			
mushrooms	1	—					13	0.1	−4.3	898	0.8	2,830,709.2§	43.9	96	0.0	2,830,709.2§	43.8
	20	9	276	1.9*	−4.2	94.1	10	2.2	−4.3	—				—			
sensit	1	—					10	6.0	−35.7	7131	269.7	2,283,951,507.5§	41.7	13	1.0	2,283,951,507.5§	41.7

Table 4 continued

Instance	k	SVM-BlockIP					CPLEX 20.1			LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%
	100	9	95	77.5*	−34.4	97.8	10	67.4	−36.7	—				—			
	1000	9	97	11.3*	−35.0	97.8	10	20.9	−36.4	—				—			
usps	1	—					14	230.1	−18.6	702	3.2	10,252,776.6§	51.9	36	0.2	10,252,776.6§	51.9
	10	14	437	21.0*	−18.7	87.7	18	21.0	−18.7	—				—			
	100	26	764	5.3	−18.9	87.6	19	17.9	−19.0	—				—			
w1a	1	—					5	1.0	0.0	371	0.1	12.8§	16.3	13	0.0	12.8§	15.2
	10	7	106	0.2*	0.0	88.8	5	0.4	0.0	—				—			
w4a	1	—					5	7.8	0.0	803	0.3	82.3§	14.4	16	0.0	82.3§	14.9
	30	7	167	0.6*	0.0	87.8	5	1.6	0.0	—				—			
w8a	1	—					6	1.0	0.0	4508	28.1	1952.7§	13.4	18	0.1	1952.7§	13.9
	200	14	580	9.9	0.1	89.2	6	32.7	0.0	—				—			
colon-cancer	1	—					5	0.0	0.0	393	0.1	0.0	64.5	150	0.0	0.0	66.1
	10	8	457	0.2*	0.0	58.1	8	0.2	−0.0	—				—			
gisette	1	—					10	713.8	2.5	556	41.0	515,140,383.9§	53.1	32	2.8	515,140,384.0§	53.1
	10	8	119	196.7*	−1417.3	88.8	10	1104.9	−346.6	—				—			
	100	13	279	40.5*	−1424.7	88.5	13	11,816.2	−246.5	—				—			
leu	1	—					7	0.1	−40.3	154	0.1	581.8§	41.2	116	0.1	581.8§	41.2
	2	6	80	0.1	−40.3	0.0	9	0.1	−40.3	—				—			
news20	1	—					12	3141.7	−0.0	1939	144.3	41,283.9§	48.8	9	1.4	41,283.9§	49.0

Table 4 continued

Instance	k	SVM-BlockIP					CPLEX 20.1			LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%
	40	10	52	109.8*	0.0	85.6	15	5586.1	−0.0	—				—			
rcv1	1	—					10	2726.6	−0.0	2288	47.0	11,331.6§	54.4	16	0.3	11,331.4§	54.4
	40	13	33	**7.4**	0.0	93.9	9	2624.1	−0.0	—				—			
	400	17	73	47.9	−0.0	88.1	11	4111.4	−0.0	—				—			
real-sim	1	—					6	71,227.9	0.0	7126	763.1	47,167.4§	33.3	16	0.7	47,167.0§	33.2
	100	11	26	**22.2**	0.0	99.6	6	79,716.8	0.0	—				—			
	1000	22	83	62.9	−0.0	90.9	10	136,660.5	−0.0	—				—			

—Particular combination of (solver,k) is either not available (for SVM-OOPS, LIBSVM, LIBLINEAR) or it was not run (for SVM-BlockIP)
*Best result with SVM-BlockIP obtained with predictor-corrector direction
§LIBSVM or LIBLINEAR stopped at a non-optimal point

Table 5 Results for one-class SVM scaled instances (in boldface the CPU of fastest execution for each instance)

Instance	k	SVM-BlockIP					CPLEX 20.1			LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%
a9a	1	—					10	**0.4**	0.0	0	5.7	32.7[§]	47.5	0	0.1	32.7[§]	47.5
	100	5	20	2.3*	0.0	100.0	12	20.3	−0.0	—				—			
	1000	6	20	0.5*	0.0	100.0	10	22.2	0.0	—				—			
australian	1	—					8	**0.1**	−0.0	0	0.0	0.0	74.2	0	0.0	0.0	74.2
	2	7	9	**0.1**	0.0	100.0	9	0.1	0.0	—				—			
covtype	1	—					5	**4.2**	0.0	11,814	1030.8	5145.3[§]	45.2	1	1.0	3973.3[§]	49.9
	1000	6	21	91.4*	0.0	100.0	5	1619.9	0.0	—				—			
	10,000	6	32	10.6*	0.0	100.0	10	73.0	0.0	—				—			
ijcnn1	1	—					7	**0.3**	0.0	0	5.5	34.7[§]	12.7	0	0.1	34.7[§]	12.7
	100	5	16	6.2*	0.0	100.0	10	20.1	−0.0	—				—			
	1000	5	25	1.0*	0.0	100.0	10	15.0	−0.0	—				—			
madelon	1	—					10	8.3	−0.0	0	0.4	2.5[§]	51.5	0	0.1	2.5[§]	51.5
	10	5	16	1.7*	0.0	100.0	8	9.5	−0.0	—				—			
mnist-ge5-lt5	1	—					9	23.5	−0.0	2499	328.5	308.3[§]	54.2	1	1.1	223.8[§]	51.0
	200	5	14	14.2*	0.0	100.0	12	533.3	−0.0	—				—			
	2000	5	18	4.5*	0.0	100.0	10	633.1	0.0	—				—			
mnist-odd-even	1	—					9	23.5	−0.0	2499	331.2	308.3[§]	60.0	1	1.1	223.8[§]	51.7
	200	5	14	14.7*	0.0	100.0	12	530.0	−0.0	—				—			
	2000	5	18	4.6*	0.0	100.0	10	632.3	0.0	—				—			
mushrooms	1	—					4	0.1	0.1	0	0.4	4.2[§]	45.8	0	0.0	4.2[§]	45.8
	20	5	17	0.9*	0.0	100.0	4	1.3	0.1	—				—			
sensit	1	—					10	5.7	0.0	0	79.1	21.1[§]	64.9	0	1.1	21.1[§]	64.9

Table 5 continued

Instance	k	SVM-BlockIP					CPLEX 20.1			LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%
	100	5	20	59.4*	0.0	100.0	12	76.4	−0.0	—				—			
	1000	5	27	7.6*	0.0	100.0	10	28.2	0.0	—				—			
usps	1	—					6	100.1	0.0	0	2.0	6.4§	52.6	0	0.2	6.4§	52.6
	10	5	20	9.3*	0.0	100.0	4	12.8	0.1	—				—			
	100	5	23	1.2*	0.0	100.0	18	22.1	−0.0	—				—			
w1a	1	—					7	1.2	0.0	0	0.0	0.1	71.8	0	0.0	0.1	71.8
	10	9	11	0.1	0.0	100.0	7	0.5	0.0	—				—			
w4a	1	—					6	8.8	0.0	0	0.3	0.4§	80.6	0	0.0	0.4§	80.6
	30	5	17	0.3*	0.0	100.0	4	1.4	0.1	—				—			
w8a	1	—					8	1.1	0.0	0	13.3	19.3§	29.5	0	0.1	19.3§	29.5
	200	5	24	2.3*	0.0	100.0	12	52.3	0.0	—				—			
colon-cancer	1	—					8	0.0	−1.6	34	0.0	61.5§	50.0	13	0.0	61.5§	37.1
	10	7	85	0.1	−1.6	90.3	6	0.2	−1.6	—				—			
gisette	1	—					8	79.5	−0.0	3	16.5	24.0§	42.0	1	0.5	12.7§	50.0
	10	5	19	6.5*	0.0	100.0	9	174.4	−0.0	—				—			
	100	5	14	1.6*	0.0	100.0	11	642.0	−0.0	—				—			
leu	1	—					8	0.1	−231.8	85	0.1	3347.9§	52.9	73	0.1	3347.9§	52.9
	2	5	42	0.1	−230.2	55.9	8	0.1	−231.8	—				—			
news20	1	—					9	2398.8	0.0	0	58.5	2.6§	43.8	0	1.5	2.6§	43.8
	40	5	22	55.2*	0.0	100.0	10	3694.7	0.0	—				—			

Table 5 continued

Instance	k	SVM-BlockIP					CPLEX 20.1			LIBSVM				LIBLINEAR			
		it	PCGit	CPU	obj	acc%	it	CPU	obj	it	CPU	obj	acc%	it	CPU	obj	acc%
rcv1	1	—					7	1972.3	0.0	0	17.3	0.7[§]	54.2	0	0.2	0.7[§]	54.2
	40	5	17	4.2*	0.0	100.0	8	2330.3	0.0	—				—			
	400	6	16	18.3*	0.0	100.0	10	3677.8	0.0	—				—			
real-sim	1	—					8	91,542.9	0.0	0	153.2	1.1[§]	32.4	0	0.6	1.1[§]	32.4
	100	5	18	15.1*	0.0	100.0	10	124,961.7	0.0	—				—			
	1000	6	27	24.6*	0.0	100.0	10	135,979.7	0.0	—				—			

—Particular combination of (solver,k) is either not available (for SVM-OOPS, LIBSVM, LIBLINEAR) or it was not run (for SVM-BlockIP)
*Best result with SVM-BlockIP obtained with predictor-corrector direction
[§]LIBSVM or LIBLINEAR stopped at a non-optimal point

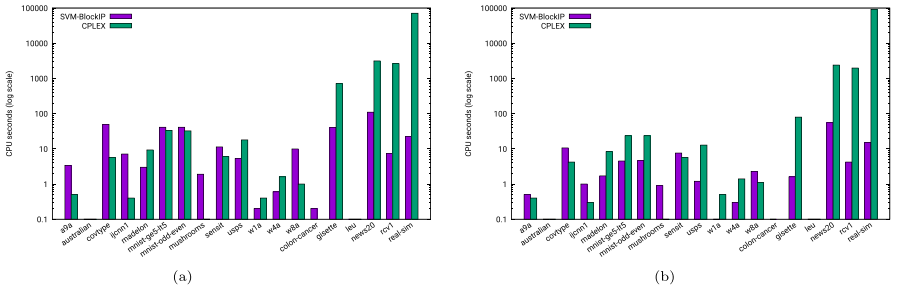

Fig. 4 CPU seconds (in log scale) required by SVM-BlockIP and CPLEX for one-class SVM, with the original instances in Table 4 (plot **a**) and the scaled instances in Table 5 (plot **b**)

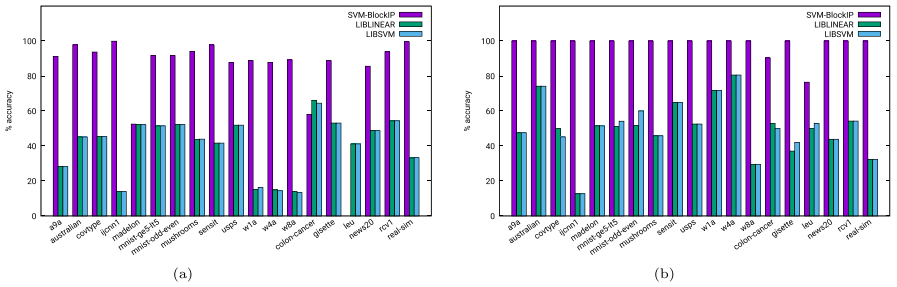

Fig. 5 Accuracies (in %) provided by solutions with SVM-BlockIP, LIBLINEAR and LIBSVM for one-class SVM, with the original instances in Table 4 (plot **a**) and the scaled instances in Table 5 (plot **b**)

CPU times for LIBSVM and LIBLINEAR are not given since they did not solve the optimization problem. It is clearly observed that SVM-BlockIP is much more efficient than CPLEX for the (rightmost) instances with a large number of features.

As for the accuracies, it can be observed in plot (a) of Fig. 5 that SVM-BlockIP generally provided values of around 90% (as expected by theory) for the runs in Table 4, except for the instances "madelon" and "colon-cancer" (in the latter it was outperformed even by LIBSVM and LIBLINEAR). For the scaled instances in Table 5, SVM-BlockIP accuracies were even higher, about 100% in most cases, as shown in plot (b) of Fig. 5. Whether SVMs with such high accuracies are useful in practice for novelty detection is a question beyond the scope of this work, which only focuses on the efficient solution of the SVM as an optimization problem.

5 Conclusions

For large-scale optimization problems arising from data science and machine learning applications, first-order coordinate descent algorithms are traditionally considered to be superior to second-order methods (in particular, to interior-point methods). For the particular case of two-class and one-class SVMs, we have shown in this work that a specialized interior-point method for an appropriate multiple variable splitting reformulation of the SVM problem can provide decent results when compared to the best

machine learning tools (i.e. LIBLINEAR). More importantly, when the optimization problem involves at least a single linear constraint (as in the dual of the one-class SVM problem), we have shown that the second-order interior-point method is very efficient and provides high-quality solutions, whereas the (far from optimal) solutions obtained by first-order algorithms (i.e. LIBSVM and LIBLINEAR) are not useful in practice. In addition, when working with high-dimensional data, the new approach presented in this work outperformed to a large degree the best interior-point methods for SVM (namely, CPLEX 20.1 and SVM-OOPS).

Acknowledgements This research has been supported by the MCIN/AEI/FEDER project RTI2018-097580-B-I00. We thank Diego Juárez for his help in the implementation of some routines of SVM-BlockIP.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Astorino, A., Fuduli, A.: Support vector machine polyhedral separability in semisupervised learning. J. Optim. Theory Appl. **164**, 1039–1050 (2015). https://doi.org/10.1007/s10957-013-0458-6
2. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. SIAM Rev. **60**, 223–311 (2018). https://doi.org/10.1137/16M1080173
3. Castro, J.: A specialized interior-point algorithm for multicommodity network flows. SIAM J. Optim. **10**, 852–877 (2000). https://doi.org/10.1137/S1052623498341879
4. Castro, J.: An interior-point approach for primal block-angular problems. Comput. Optim. Appl. **36**, 195–219 (2007). https://doi.org/10.1007/s10589-006-9000-1
5. Castro, J.: Interior-point solver for convex separable block-angular problems. Optim. Methods Softw. **31**, 88–109 (2016). https://doi.org/10.1080/10556788.2015.1050014
6. Castro, J., Cuesta, J.: Quadratic regularizations in an interior-point method for primal block-angular problems. Math. Program. **130**, 415–445 (2011). https://doi.org/10.1007/s10107-010-0341-2
7. Chang, C.-C., Lin, C.-J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**, 1–27 (2011). https://doi.org/10.1145/1961189.1961199
8. Chou, H.-Y., Lin, P.-Y., Lin, C.-J.: Dual coordinate-descent methods for linear one-class SVM and SVDD. In: Proceedings of the 2020 SIAM International Conference on Data Mining, pp. 181–189 (2020). https://doi.org/10.1137/1.9781611976236.21
9. Cortes, C., Vapnik, V.: Support vector networks. Mach. Learn. **20**, 273–297 (1995). https://doi.org/10.1007/BF00994018
10. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, Cambridge (2000). https://doi.org/10.1017/CBO9780511801389
11. Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., Lin, C.-J.: LIBLINEAR: a library for large linear classification. J. Mach. Learn. Res. **9**, 1871–1874 (2008)
12. Ferris, M., Munson, T.: Interior point methods for massive support vector machines. SIAM J. Optim. **13**, 783–804 (2003). https://doi.org/10.1137/S1052623400374379
13. Gertz, E.M., Griffin, J.D.: Support vector machine classifiers for large data sets. Argonne National Laboratory, Technical Report ANL/MCS-TM-289. https://www.osti.gov/biblio/881587 (2005)

14. Goldfarb, D., Scheinberg, K.: A product-form Cholesky factorization method for handling dense columns in interior point methods for linear programming. Math. Program. **99**, 1–34 (2004). https://doi.org/10.1007/s10107-003-0377-7
15. Goldfarb, D., Scheinberg, K.: Solving structured convex quadratic programs by interior point methods with application to support vector machines and portfolio optimization. IBM Research Report RC23773 (W0511-025) (2005)
16. Gondzio, J.: Convergence analysis of an inexact feasible interior point method for convex quadratic programming. SIAM J. Optim. **23**, 1510–1527 (2013). https://doi.org/10.1137/120886017
17. Mészáros, C.: On free variables in interior point methods. Optim. Methods Softw. **9**, 121–139 (1998). https://doi.org/10.1080/10556789808805689
18. Ortega, J.M.: Introduction to Parallel and Vector Solutions of Linear Systems. Frontiers of Computer Science, Springer, Boston (1988). https://doi.org/10.1007/978-1-4899-2112-3
19. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: Schölkopf, B., Burges, C.J.C., Smola, A.J. (eds.) Advances in Kernel Methods: Support Vector Learning, 185–20. MIT Press, Cambridge (1999)
20. Schölkopf, B., Platt, J.C., Shawe-Taylor, J., Smola, A.J.: Estimating the support of a high-dimensional distribution. Neural Comput. **13**, 1443–1471 (2001). https://doi.org/10.1162/089976601750264965
21. Woodsend, K., Gondzio, J.: Exploiting separability in large-scale linear support vector machine training. Comput. Optim. Appl. **49**, 241–269 (2011). https://doi.org/10.1007/s10589-009-9296-8
22. Wright, S.J.: Primal-Dual Interior-Point Methods. SIAM, Philadelphia (1997). https://doi.org/10.1137/1.9781611971453
23. Wright, S.J.: Coordinate descent algorithms. Math. Program. **151**, 3–34 (2015). https://doi.org/10.1007/s10107-015-0892-3