J. Nonlinear Var. Anal. 10 (2026), No. 2, pp. 363-379
Available online at http://jnva.biemdas.com
https://doi.org/10.23952/jnva.10.2026.2.07

A RANK-ONE-UPDATE METHOD FOR THE TRAINING OF SUPPORT VECTOR MACHINES

FLORIAN JARRE

Faculty of Mathematics and Natural Sciences, Heinrich Heine Universität Düsseldorf, Germany

Abstract. This paper considers convex quadratic programs associated with the training of support vector machines (SVMs). Exploiting the special structure of the SVM problem, a new type of active set method with long cycles and stable rank-one-updates is proposed and tested (CMU: cycling method with updates). The structure of the problem allows for a repeated simple increase of the set of inactive constraints while controlling its size. This is followed by minimization steps with cheap updates of a matrix factorization. A widely used approach for solving SVM problems is the alternating direction method SMO, a method that is very efficient for generating low accuracy solutions. The new active set approach allows for higher accuracy results at moderate computational cost. To relate both approaches, the effect of the accuracy on the running time and on the predictive quality of the SVM is compared based on some numerical examples. A surprising result of the numerical examples is that only a very small number of cycles (each consisting of less than $2n$ steps) was used for CMU.

Keywords. Active set method; Rank-one-update; Support vector machine.

1. Introduction

An active set descent algorithm is proposed for solving the problem of training a support vector machine (SVM) by exploiting the special structure of the problem and solving it with a sequence of *cycles*. Each cycle begins with an *up-cycle* consisting of a repeated increase of an *inactive set* using very cheap first-order descent steps. This increase is followed by a *down-cycle* or *sweep* of eliminating *inactive indices* one by one, where also each Newton step in the sweep is computationally cheap using rank-one-updates. Only at the beginning of each sweep a Cholesky factorization of the *inactive part of the Hessian* is computed. The overall method is referred to as cycling method with updates (CMU).

A rank-one-update would also be possible in the up-cycle while increasing the active set and would save the Cholesky factorization at the beginning of a sweep. However, on the one side, increasing the dimension of the Cholesky factorization is more sensitive to numerical rounding errors than decreasing it. On the other side, in the up-cycle, the factorization of the Hessian matrix is not needed for the repeated increases of the active set, and the overall numerical cost of a full cycle is comparable to the cost that would have been applicable when relying on (many) rank-one-updates in the up-cycle as well. Moreover, the numerical experiments suggest that for

E-mail address: jarre@hhu.de.

Received 8 April 2025; Accepted 31 August 2025; Published online 1 January 2026.

many problems only a very small number of cycles is necessary, less than 10 in all examples that were tested.

In many situations, the method of choice for the training of SVMs is an alternating direction method referred to as Sequential Minimal Optimization (SMO), [1], that generates approximate solutions of moderate accuracy in very short time.

On the other hand, the danger of overfitting the SVM is usually controlled by the parameters of the kernel and of the so-called *soft margin.* Solving the SVM problem to high accuracy generally does not lead to overfitting when the parameters are selected properly. If the accuracy needed for a given application is not known in advance, a higher accuracy solution may allow to extract more information from a given training set. Some simple numerical examples that illustrate the benefit of a higher numerical accuracy are presented in Section 4, where the active set method of this paper, CMU is compared with a variant of SMO for selected problems, concentrating on the number of arithmetic operations, final accuracy, and predictive quality.

The setting of SVMs in the present paper follows the outline in [2]; for further discussions of SVMs, we refer to [3, 4, 5, 6] and the references given there. For SVMs with kernels as considered here, the so-called feature space (that replaces the data space for the classification task; see, for example, [3, 4, 5, 6, 2]) may be infinite dimensional, and if not, it is typically high-dimensional. This implies that for such kernels the low-rank-update formulas that are the basis of extremely efficient interior-point solvers for SVMs with low-dimensional data spaces such as presented in [7, 8] cannot be applied. While for SVMs as considered in [7, 8] the number of data points may be large, 10^5 or even 10^6, the method in [7, 8] does not apply to Gaussian or other kernels as considered in this paper. The method of the present paper is not suitable for much more than 10^4 data points.

1.1. Notation.

$A \succeq 0$ indicates that A is a symmetric positive semidefinite matrix, and $A \succ 0$ denotes (strict) positive definiteness. For a vector $x \in \mathbb{R}^n$ the inequality $x \geq 0$ applies component-wise. Given two vectors $x, y \in \mathbb{R}^n$, their Hadamard product (component-wise product) is denoted by $x \circ y \in \mathbb{R}^n$. The vector $e := (1, 1, \ldots, 1)^T$ always denotes the all-one-vector with its dimension given by the context. The vector e_i is the i-th canonical unit vector. Let $\mathbf{I}, \mathbf{J} \subset \{1, \ldots, n\}$, a vector $x \in \mathbb{R}^n$, and a matrix $H \in \mathbb{R}^{n \times n}$ be given. The vector $x_{\mathbf{I}} \in \mathbb{R}^{|\mathbf{I}|}$ is the vector with components x_i for $i \in \mathbf{I}$ and $H_{\mathbf{I},\mathbf{J}} \in \mathbb{R}^{|\mathbf{I}| \times |\mathbf{J}|}$ denotes the submatrix of H with entries $H_{i,j}$ for $i \in \mathbf{I}$ and $j \in \mathbf{J}$.

1.2. Basic problem in the training of support vector machines.

The problem that is to be solved for "training" an SVM with kernel and with soft margins is a convex quadratic program of the form

$$\min_{x \in \mathbb{R}^n} \left\{ \frac{1}{2} x^T H x - c^T x \;\middle|\; z^T x = 0,\ Ce \geq x \geq 0 \right\}, \tag{1.1}$$

where $H \succ 0$ generally has the form $H = ZKZ$. Here, K is a *kernel matrix* and Z a diagonal matrix, $C > 0$ is a constant associated with the penalty term of the soft margin, $z \in \{\pm 1\}^n$ is the vector of *labels* for the classification, and $c = e$ in general. Throughout we assume that the SVM-problem is well-posed in the sense that $z \notin \{-e, e\}$.

The objective function is abbreviated as

$$q(x) \equiv \frac{1}{2} x^T H x - c^T x,$$

the box $[0,C]^n$ is denoted by $\mathscr{B} := [0,C]^n$, and the null space given by the equality constraint is denoted by $\mathscr{N} := \{ x \mid z^T x = 0 \}$. Thus, the feasible set of (1.1) is $\mathscr{B} \cap \mathscr{N}$. For later use, we note that the orthogonal projection onto $\mathscr{N}$ is given by $\Pi_{\mathscr{N}} := I - zz^T/n$ because $z^T z = n$. For a feasible point x and some vector $y \in \mathbb{R}^n$, the projection of y onto the tangential cone of $\mathscr{B}$ at x is defined as $p := \Pi_{\mathscr{B},x}(y)$ with components

$$p_i := \begin{cases} \max\{0, y_i\} & \text{if } x_i = 0, \\ y_i & \text{if } x_i \in (0,C), \\ \min\{0, y_i\} & \text{if } x_i = C. \end{cases} \tag{1.2}$$

1.3. **Optimality conditions.** In Proposition 1.1 below, the standard optimality conditions for (1.1) are formulated in a simple form that is used for the definition of the CMU algorithm.

For a given point $x \in \mathscr{B}$, the set of *active indices or active inequalities* at x is always denoted as

$$\mathbf{A} := \{i \mid x_i = 0\} \cup \{i \mid x_i = C\}. \tag{1.3}$$

The set of *inactive indices* at a given point x is always denoted as $\mathbf{K} := \{1, \ldots, n\} \backslash \mathbf{A}$.

Denote the gradient of q at $x \in \mathbb{R}^n$ by $g := g(x) := Hx - c$. For $x \in \mathscr{B}$, let $\sigma := \sigma(x) \in \mathbb{R}^n$ be defined as

$$\sigma_i := \begin{cases} 1 & \text{if } x_i = 0, \\ 0 & \text{if } x_i \in (0,C), \\ -1 & \text{if } x_i = C. \end{cases} \tag{1.4}$$

Then a direction s at a point $x \in \mathscr{B} \cap \mathscr{N}$ is a *feasible direction* at x (i.e., $x + \lambda s \in \mathscr{B} \cap \mathscr{N}$ for small $\lambda > 0$) if and only if $z^T s = 0$ and $\sigma \circ s \geq 0$. The following proposition formulates the optimality conditions for (1.1).

Proposition 1.1. *Let $x \in \mathscr{B} \cap \mathscr{N}$, $g = Hx - c$, and*

$$\mu := \begin{cases} g_{\mathbf{K}}^T z_{\mathbf{K}} / |\mathbf{K}| & \textit{if } \mathbf{K} \neq \emptyset, \\ \min\{\sigma_i g_i \mid i : \ \sigma_i z_i = 1\} & \textit{else}. \end{cases}$$

Set $\tilde{g} := g - \mu z$. Then x is an optimal solution of (1.1) if and only if $\sigma \circ \tilde{g} \geq 0$ and $\tilde{g}_{\mathbf{K}} = 0$. (Here, $0 \in \mathbb{R}^{|\mathbf{K}|}$ and for $\mathbf{K} = \emptyset$ the condition $\tilde{g}_{\mathbf{K}} = 0$ is trivially satisfied.)

Proof. The conditions of Proposition 1.1 can be rewritten such that they are equivalent to the standard (necessary and sufficient) KKT conditions. The proof below is a bit longer, and in case that the conditions are violated it derives descent directions that are used in the algorithm.

"$\Rightarrow$" ($\mathbf{K} \neq \emptyset$): Let $x \in \mathscr{B} \cap \mathscr{N}$ be an optimal solution of (1.1) with $\mathbf{K} \neq \emptyset$. Then the standard KKT conditions imply that there exists $\mu \in \mathbb{R}$ such that

$$g_{\mathbf{K}} = \mu z_{\mathbf{K}}, \tag{1.5}$$

i.e., $z_k g_k = \mu$ for all $k \in \mathbf{K}$ (because $z_k^2 = 1$). Setting $\mu := g_{\mathbf{K}}^T z_{\mathbf{K}} / |\mathbf{K}|$ and $\tilde{g} := g - \mu z$, relation (1.5) is equivalent to $\tilde{g}_{\mathbf{K}} = 0$.

Now, we assume that there exists an index $i \in \mathbf{A}$ with $\sigma_i g_i < \sigma_i z_i \mu$. Then, for some $k \in \mathbf{K}$, we define $s := \sigma_i e_i - \sigma_i z_i z_k e_k$. It follows that $z^T s = \sigma_i z_i - \sigma_i z_i z_k^2 = 0$ and $\sigma \circ s = \sigma_i^2 e_i - \sigma_i \sigma_k z_i z_k e_k \geq 0$ since $\sigma_k = 0$. Hence, s is a feasible direction at x. Moreover, $g^T s = \sigma_i g_i - \sigma_i z_i z_k g_k = \sigma_i g_i - \sigma_i z_i \mu < 0$ in contradiction to the optimality of x. Hence, it follows that

$$\sigma_i g_i \geq \sigma_i z_i \mu \quad \forall i \in \mathbf{A} \qquad \text{i.e.} \qquad \sigma_{\mathbf{A}} \circ g_{\mathbf{A}} \geq \mu \sigma_{\mathbf{A}} \circ z_{\mathbf{A}} \qquad \text{or} \qquad \sigma_{\mathbf{A}} \circ (g_{\mathbf{A}} - \mu z_{\mathbf{A}}) \geq 0. \tag{1.6}$$

Since $\sigma_{\mathbf{K}} = 0$ relation (1.6) is the same as

$$\sigma \circ (g - \mu z) = \sigma \circ \tilde{g} \geq 0. \tag{1.7}$$

"$\Leftarrow$": Conversely, let (1.7) be satisfied and a feasible direction $\bar{s}$ minimizing $g^T s$ be given,

$$\bar{s} \in \operatorname{argmin}\{g^T s \mid z^T s = 0, \ \ \sigma \circ s \geq 0, \ \ \|s\|_2 \leq 1\}.$$

Using (1.7) and $\tilde{g}_{\mathbf{K}} = 0$, one has

$$g^T \bar{s} = (g - \mu z)^T \bar{s} = \tilde{g}^T \bar{s} = (\sigma \circ \tilde{g})^T (\sigma \circ \bar{s}) \geq 0.$$

Hence, there is no feasible descent direction starting at x and thus (by convexity and linearity of the constraints), x is a minimizer of (1.1).

"$\Rightarrow$" ($\mathbf{K} = \emptyset$): Similarly, let $x \in \mathscr{B} \cap \mathscr{N}$ be an optimal solution of (1.1) with $\mathbf{K} = \emptyset$. Then (1.5) is void. Condition (1.7), i.e., the inequality "$\sigma \circ \tilde{g} \geq 0$" is equivalent to

$$\min\{\sigma_i g_i \mid i: \ \sigma_i z_i = 1\} := \mu \geq \max\{-\sigma_k g_k \mid k: \ \sigma_k z_k = -1\}, \tag{1.8}$$

where the case that $\sigma_i z_i = 1$ for all $1 \leq i \leq n$ or $\sigma_k z_k = -1$ for all $1 \leq k \leq n$ cannot occur so that the min and max are well defined. (If, for example, $\sigma_i z_i = 1$ for all i, then, for each i, either $\sigma_i = z_i = 1$, in which case $x_i = 0$, or $\sigma_i = z_i = -1$, in which case $x_i = C > 0$. And then, $x^T z = \sum_{i: z_i = -1} C z_i < 0$ since $z \notin \{-e, e\}$. Thus, x is not feasible for (1.1).)

Indeed, when $\mathbf{K} = \emptyset$, let i be an index with $\sigma_i z_i = 1$ and $\sigma_i g_i = \mu$, and assume that there exists an index k with $\sigma_k z_k = -1$ and $\mu < -\sigma_k g_k$. Then

$$s := \sigma_i e_i - \sigma_i z_i z_k e_k = \sigma_i e_i - z_k e_k = \sigma_i e_i + \sigma_k e_k$$

satisfies $z^T s = \sigma_i z_i - \sigma_i z_i z_k^2 = 0$ and $\sigma \circ s \geq 0$. Hence, s is a feasible direction at x. Moreover, $g^T s = \sigma_i g_i - \sigma_i z_i z_k g_k = \mu - z_k g_k = \mu + \sigma_k g_k < 0$ in contradiction to the optimality of x. Hence, (1.8) must hold at an optimal solution x with $\mathbf{K} = \emptyset$.

"$\Leftarrow$": Conversely, (1.8) implies $-\mu \leq \min\{\sigma_k g_k \mid \sigma_k z_k = -1\}$ i.e., $\sigma_k g_k \geq -\mu$ for k with $\sigma_k z_k = -1$. Using the definition of μ in (1.8), this implies $\sigma_i g_i \geq \mu \sigma_i z_i$ for all $i \in \{1, \ldots, n\}$ or $\sigma \circ (g - \mu z) \geq 0$. Given a feasible direction $\bar{s}$ minimizing $g^T s$,

$$\bar{s} \in \operatorname{argmin}\{g^T s \mid z^T s = 0, \ \ \sigma \circ s \geq 0, \ \ \|s\|_2 \leq 1\},$$

using $\mathbf{K} = \emptyset$, i.e., $\sigma_i^2 = 1$ for all i, it follows that

$$g^T \bar{s} = (g - \mu z)^T \bar{s} = ((g - \mu z) \circ \sigma)^T (\sigma \circ \bar{s}) \geq 0.$$

Hence, there is no feasible descent direction starting at x. Thus x is a minimizer of (1.1). □

Summarizing, the necessary and sufficient conditions for optimality of a feasible point x are also given by (1.5), (1.7), and (1.8).

2. An Active Set Rank-One-Update Algorithm

2.1. **Outline.** From a theoretical point of view, problem (1.1) is well understood and many globally convergent algorithms are available even for more general convex quadratic programs. From a practical point of view in turn, the exploitation of the given structure matters for reducing the overall computational effort, while generating a solution with sufficiently high accuracy. In this respect, the method of choice in many situations is the alternating direction method

SMO, [1]. The present paper is an attempt to improve over SMO in certain other situations, in particular, when a numerical solution of high accuracy is needed.

The CMU method proposed in this paper for solving (1.1) always generates feasible iterates. It is divided into inner iterations and outer iterations: Each outer iteration consists of a sweep reducing the set of inactive variables, followed by an up-cycle increasing this set again. More precisely, each sweep starts at an initial point x in the box $\mathscr{B}$ satisfying $z^T x = 0$. The sweep uses Newton's method to successively remove inactive indices one by one using cheap updates of the Cholesky factor. When no further inactive indices can be removed, the up-cycle begins, adding again a moderate number of active indices to the inactive set. All steps are such that the objective function decreases.

In more detail, a sweep starting at a point x is as follows: First, a factorization of the inactive part of the Hessian is computed. Then, while keeping the active variables fixed at their current value a Newton step s starting at x is computed for minimizing q while maintaining the equality $z^T s = 0$ and while changing only the inactive variables. If the result of the Newton step $x^+ := x+s$ satisfies $x_i^+ \leq 0$ or $x_i^+ \geq C$, the step length of the Newton step is reduced so that x^+ lies at the boundary of the box $\mathscr{B}$, and the set of inactive variables is reduced accordingly. After removing an inactive index, the Cholesky factor is updated. Then the Newton iterations are restarted. This is repeated until the Newton iterations result in a step that does not lead to a reduction of the set of inactive indices any more. A key point of this sweep is that the update of the Cholesky factor for the Newton step can be carried out with order n^2 operations for each inactive variable that is removed. (In general, the Cholesky factor for this application will not be sparse, but if the Cholesky factor happens to be sparse, there also exist updates that exploit sparsity; see, for example, [9].)

When no further variable becomes active while performing the Newton step, the up-cycle begins. At each step of the up-cycle, a feasible descent step for q is computed that turns at least one of the active variables to be inactive. The step of adding inactive variables is repeated as long as possible, but only until the active set has increased by at most 50%. This way a moderate number of new inactive variables are generated before the next sweep is started. The upper bound of 50% is intended to keep the size of the Cholesky factor moderately small. If the up-cycle fails at the very first attempt to turn one or two of the active variables to be inactive, the overall algorithm is terminated.

In what follows, the above steps are detailed while observing the computational cost, numerical accuracy, and global convergence properties.

2.2. **Up-cycle, leaving active constraints.** A key feature of CMU is that in fairly general situations as detailed in this subsection, iterates can easily be moved away from parts of the boundary of $\mathscr{B}$ while decreasing the objective function q. This allows the repeated increase of the set of inactive indices before carrying out a numerically expensive recomputation of a Cholesky factor.

The following steps are somewhat "technical" but numerically very cheap.

Let a feasible point x for (1.1) be given and assume that there is at least one active inequality at x, i.e., $\mathbf{A} \neq \emptyset$. In order to reduce the set of active inequalities, i.e., to find a feasible point x^+ with lower objective function value and with a smaller set of active inequalities the following procedure is used.

Define the gradient of q at x as $g := Hx - c$. A descent step $\widetilde{s}$ is then defined as follows.

If the set of inactive indices $\mathbf{K}$ is not empty, let $\mu := g_{\mathbf{K}}^T z_{\mathbf{K}}/|\mathbf{K}|$ and $\widetilde{g} := g - \mu z$ (see relation (1.5)). Else, let μ be as defined in in Proposition 1.1. Then set

$$\widetilde{g} := g - \mu z \qquad \text{and} \qquad \widetilde{s} := \Pi_{\mathscr{B},x}(-\widetilde{g}). \tag{2.1}$$

Then, $\widetilde{g}^T\widetilde{s} \le 0$ and for $\widetilde{s} \ne 0$ the inequality is strict. Furthermore, $\widetilde{s}$ is a feasible direction with respect to all inequality constraints, i.e. $x + \lambda\widetilde{s} \in \mathscr{B}$ for small $\lambda > 0$, but in general, $z^T\widetilde{s} \ne 0$. The next steps below and in Section 2.2.1 aim at modifying $\widetilde{s}$ to a feasible descent direction.

Any vector $s \in \mathscr{N}$ satisfies $z^T s = 0$ and

$$g^T s = (g - \mu z)^T s = \widetilde{g}^T s. \tag{2.2}$$

Therefore $s \in \mathscr{N}$ is a descent direction for q at x whenever $\widetilde{g}^T s < 0$.

Let $\hat{s}$ be a feasible direction for q starting at x. Then, $\hat{s}$ must satisfy $\hat{s} \in \mathscr{N}$ as well as $\hat{s} = \Pi_{\mathscr{B},x}(\hat{s})$. It follows from (2.2) that $g^T\hat{s} = \widetilde{g}^T\hat{s} \ge -(\Pi_{\mathscr{B},x}(-\widetilde{g}))^T\hat{s} = -\widetilde{s}^T\hat{s}$, where the inequality follows from relation (2.3) below:

- Let x be feasible for (1.1), and let $\hat{s}$ be given with $\hat{s} = \Pi_{\mathscr{B},x}(\hat{s})$. Then, for arbitrary g, it follows that
$$g^T\hat{s} \ge -(\Pi_{\mathscr{B},x}(-g))^T\hat{s}. \tag{2.3}$$
Indeed, let $I := \{i \mid x_i = 0\}$, $J := \{j \mid x_j \in (0, C)\}$, and $K := \{k \mid x_k = C\}$. Then, $\hat{s}_I \ge 0$ and $\hat{s}_K \le 0$. Let $p := \Pi_{\mathscr{B},x}(-g)$. Then $p_I = \max(0, -g_I) \ge -g_I$ and $p_K = \min(0, -g_K) \le -g_K$, where min and max are taken componentwise, and
$$-g^T\hat{s} = \underbrace{-g_I^T}_{\le p_I^T}\underbrace{\hat{s}_I}_{\ge 0} + \underbrace{-g_J^T}_{= p_J^T}\hat{s}_J + \underbrace{-g_K^T}_{\ge p_K^T}\underbrace{\hat{s}_K}_{\le 0} \le p_I^T\hat{s}_I + p_J^T\hat{s}_J + p_K^T\hat{s}_K = p^T\hat{s}.$$
Therefore,
$$g^T\hat{s} \ge -p^T\hat{s} = -(\Pi_{\mathscr{B},x}(-g))^T\hat{s}.$$
□

Hence, any feasible descent direction $\hat{s}$ starting at $x \in \mathscr{B} \cap \mathscr{N}$ satisfies $0 > g^T\hat{s} \ge -\widetilde{s}^T\hat{s}$. In particular, the following proposition is true.

Proposition 2.1. *If x is feasible for (1.1), $g := Hx - c$, and $\widetilde{s}$ given by (2.1) satisfies $\widetilde{s} = 0$, then there does not exist a feasible descent direction, i.e., x solves the convex problem (1.1).*

2.2.1. **Computing a search direction:** In the following, for a feasible point x, let $\widetilde{s} \ne 0$ be as in (2.1) and let

$$\mathbf{I} := \{i \in \mathbf{A} \mid z_i\widetilde{s}_i > 0\} \quad \text{and} \quad \mathbf{J} := \{i \in \mathbf{A} \mid z_i\widetilde{s}_i < 0\}$$

be the sets of active indices that increase/decrease the constraint term $z^T\widetilde{s}$. For defining a feasible descent direction that makes certain active variables inactive, the following cases are considered:

Case 1. If $\mathbf{I}$ and $\mathbf{J}$ are both nonempty, we define a feasible direction s as follows:

$$v_1 := z_{\mathbf{I}}^T\widetilde{s}_{\mathbf{I}} > 0, \quad v_2 := z_{\mathbf{J}}^T\widetilde{s}_{\mathbf{J}} < 0, \quad s_{\mathbf{I}} := -v_2\widetilde{s}_{\mathbf{I}}, \quad s_{\mathbf{J}} := v_1\widetilde{s}_{\mathbf{J}}, \tag{2.4}$$

and $s_i := 0$ for all i not in $\mathbf{I} \cup \mathbf{J}$. Then,

$$z^T s = z_{\mathbf{I}}^T s_{\mathbf{I}} + z_{\mathbf{J}}^T s_{\mathbf{J}} = -v_2 z_{\mathbf{I}}^T\widetilde{s}_{\mathbf{I}} + v_1 z_{\mathbf{J}}^T\widetilde{s}_{\mathbf{J}} = -v_2 v_1 + v_1 v_2 = 0$$

and $\widetilde{g}^T s < 0$ since all nonzero components of s have opposite sign of the associated components in $\widetilde{g}$. Moreover, by the sign structure of s, it is a feasible descent direction that has at least two nonzero active components. So, at least two active components become inactive along the line $x+\lambda s$ for $\lambda > 0$, and large values of λ are possible without violating any inequality constraint since s only moves active indices "away from the boundary" (see also the last paragraph in Section 2.7).

Case 2: When exactly one of the sets $\mathbf{I}$ or $\mathbf{J}$ is empty, let $i \in \mathbf{A}$ be an active index with maximal value of $|\widetilde{s}_i|$. By definition of $\mathbf{I}$ and $\mathbf{J}$, it follows that $|\widetilde{s}_i| > 0$ and $\operatorname{sign}(\widetilde{s}_i) = \sigma_i = -\operatorname{sign}(\widetilde{g}_i)$.

Let

$$\overline{\mathbf{K}} := \{j \mid \sigma_i \sigma_j z_i z_j \le 0\}. \tag{2.5}$$

Since $\sigma_j = 0$ for $j \in \mathbf{K}$, it follows that $\mathbf{K} \subset \overline{\mathbf{K}}$. In addition, $\overline{\mathbf{K}}$ contains active indices j such that the vector

$$s := \operatorname{sign}(\widetilde{s}_i)(e_i - z_i z_j e_j) \tag{2.6}$$

is a feasible direction with respect to all constraints of (1.1). Indeed, $z^T s = \operatorname{sign}(\widetilde{s}_i)(z_i - z_i z_j^2) = 0$ and $\sigma \circ s = \sigma_i^2 e_i - \sigma_i \sigma_j z_i z_j e_j \ge 0$.

The linearization of the objective function along the above search direction is

$$g^T s = \widetilde{g}^T s = \widetilde{g}^T \operatorname{sign}(\widetilde{s}_i)(e_i - z_i z_j e_j) = \widetilde{g}_i \operatorname{sign}(\widetilde{s}_i) - \operatorname{sign}(\widetilde{s}_i) z_i z_j \widetilde{g}_j = -|\widetilde{g}_i| - \operatorname{sign}(\widetilde{s}_i) z_i z_j \widetilde{g}_j,$$

since $\operatorname{sign}(\widetilde{g}_i) = -\operatorname{sign}(\widetilde{s}_i)$. Therefore, an inactive index j is selected with maximum value of $\operatorname{sign}(\widetilde{s}_i) z_i z_j \widetilde{g}_j$ in order to define s. The maximum value may be negative, and if it is less or equal to $-|\widetilde{g}_i|$, the resulting s is not a descent direction. In this case, the increase of the inactive set fails and the up-cycle is terminated.

Case 3. If $\mathbf{I} = \mathbf{J} = \emptyset$, then the increase of the inactive set fails as well and the up-cycle is terminated.

Remark 2.1. Assume that the up-cycle fails at the first step immediately after the sweep with Newton iterations is completed. Let x be the iterate generated by the last Newton iteration and denote the sets of active and inactive indices at x by $\mathbf{A}$ and $\mathbf{K}$, respectively. By Newton's method, either $\mathbf{K} = \emptyset$ or $\widetilde{g}_{\mathbf{K}} = 0$. Therefore, if failure happens in Case 3, then $\widetilde{s} = 0$, and by Proposition 2.1, x is an optimal solution of (1.1). If $\mathbf{K}$ is not empty, then, since $\widetilde{g}_{\mathbf{K}} = 0$, the failure in the up-cycle cannot happen in Case 2. Hence in this case as well, x is an optimal solution of (1.1). When $\mathbf{K} = \emptyset$, i.e., when $\mathbf{A} = \{1, \dots, n\}$ failure could happen in Case 2. In this case, by definition of i, the value $\sigma_i g_i$ either coincides with the "min" in (1.8) or with the "max", and by the criterion that leads to the failure, the inequality in (1.8) is violated. Again, it follows that x is a minimizer of (1.1).

2.2.2. *Line search.* If a feasible descent direction s was found in Case 1 or Case 2, a line search is carried out along the direction s minimizing $q(x+\lambda s)$ for $\lambda \ge 0$, i.e., first define the exact line search step length

$$\hat{\lambda} := -\nabla q(x)^T s / (s^T H s) > 0$$

and the maximum feasible step length along λs,

$$\lambda_{max} := \min\left\{ \min_{i: s_i > 0} \{(C - x_i)/s_i\}, \;\; \min_{i: s_i < 0} \{-x_i/s_i\} \right\}.$$

Then, we let $\lambda := \min\left\{ \hat{\lambda}, \lambda_{max} \right\}$ and define the point

$$x^+ := x + \lambda s. \tag{2.7}$$

If $\lambda = \lambda_{max}$, a new active constraint is added while at least one other active constraint becomes inactive. In principle, this might possibly lead to cycling leaving and adding the same constraints along shorter and shorter steps. Therefore the up-cycle is limited to at most n steps before starting another sweep.

Summarizing, the up-cycle is as follows.

2.2.3. *Summary, up-cycle.* Let $\mathbf{A} \neq \emptyset$ be as in (1.3), and define $\widetilde{s}$ as in (2.1).
Set max$\mathbf{A} := \min\{n, \max\{100, \lceil 3|A|/2 \rceil\}\}$, $k := 0$.
Set $\mathbf{I} := \{i \in \mathbf{A} \mid z_i\widetilde{s}_i > 0\}$ and $\mathbf{J} := \{i \in \mathbf{A} \mid z_i\widetilde{s}_i < 0\}$.
Repeat Step 1. – Step 3. until a Stop command is encountered:

(1) **Case 1.** If **I** and **J** are both nonempty, define s as in (2.4).
Case 2. If exactly one of the sets **I** or **J** is nonempty, let i maximize $|\widetilde{s}_i|$ for i in **A**. Then j is selected as to maximize $\text{sign}(\widetilde{s}_i)z_iz_j\widetilde{g}_j$ for all indices j in $\overline{\mathbf{K}}$ defined in (2.5). Then define s as in (2.6).
Case 3. If both the sets **I** or **J** are empty, set $s = 0$.

(2) If s generated in Step 1 is not a (strict) descent step, Stop. The further reduction of the set of active indices fails.
Else, define the next iterate $x := x^+$ with x^+ given in (2.7). Update **A**, **I**, **J**, and set $k := k + 1$. If $k \geq n$ or if $|A| \geq$ max**A**, Stop.

Remark 2.2. If the step length in the up-cycle leads to the boundary of $\mathscr{B}$, then the cardinality of **A** might not increase for this step. To avoid discussions of possible cycling when this case occurs repeatedly, the safeguard query "If $k \geq n$" is added in Step 3 above. (And the restriction, not to force max**A** below 100 is subject to change; it is merely intended to reduce the number of cycles.)

2.3. **Starting point.** To find a suitable starting point for the overall algorithm, the following procedure is used. Set $s := \Pi_{\mathscr{N}}c$. As $c = e$ and $z \in \{\pm 1\}^n \backslash \{-e, e\}$, it follows that $|z^Tc| < z^Tz$ and $s > 0$. Then do a line search as in Section 2.2.2 minimizing q along $q(0 + \lambda s)$. When $\lambda < \lambda_{max}$, the result of the line search defines a point where all variables are inactive. In this case, when n is large, the computational effort for computing a Cholesky factor at this point will be large. To limit the cost of the Cholesky factor, another point is chosen as starting point for CMU. From the above point λs determine a set of indices of moderate cardinality that are "promising" (with the largest values $x_i - (\Pi_{\mathscr{N}}\nabla q(x))_i$) and define a search step s using only these promising coordinates projected onto $\{s \mid z^Ts = 0\}$. Then repeat the line search as detailed above.

2.4. **Down-cycle, Newton iterations.** Let a feasible point x with set of active indices **A** be given. Let **K** be the set of the remaining (i.e., the inactive) indices. Keeping $x_{\mathbf{A}}$ fixed in $x = (x_{\mathbf{A}}^T, x_{\mathbf{K}}^T + \Delta x_{\mathbf{K}})^T$, the minimization of q with respect to $\Delta x_{\mathbf{K}}$ subject to $z_{\mathbf{K}}^T\Delta x_{\mathbf{K}} = 0$ is considered next. As described in Section 1.2, the objective function is given by

$$q(x) = \frac{1}{2}x^THx - c^Tx = \frac{1}{2}(x_{\mathbf{A}}^T, x_{\mathbf{K}}^T + \Delta x_{\mathbf{K}}^T)H(x_{\mathbf{A}}^T, x_{\mathbf{K}}^T + \Delta x_{\mathbf{K}}^T)^T - c^T(x_{\mathbf{A}}^T, x_{\mathbf{K}}^T + \Delta x_{\mathbf{K}}^T)^T$$

$$= \frac{1}{2}\Delta x_{\mathbf{K}}^T H_{\mathbf{K},\mathbf{K}}\Delta x_{\mathbf{K}} - (c_{\mathbf{K}} - H_{\mathbf{K},\mathbf{A}}x_{\mathbf{A}} - H_{\mathbf{K},\mathbf{K}}x_{\mathbf{K}})^T\Delta x_{\mathbf{K}} + const$$

$$= \frac{1}{2}\Delta x_{\mathbf{K}}^T H_{\mathbf{K},\mathbf{K}} \Delta x_{\mathbf{K}} - (c - Hx)_{\mathbf{K}}^T \Delta x_{\mathbf{K}} + const,$$

where *const* is a term that does not depend on $\Delta x_{\mathbf{K}}$. The Newton step for minimizing q within the set $\mathscr{N}$ is given by the linear system

$$H_{\mathbf{K},\mathbf{K}}\Delta x_{\mathbf{K}} + \eta z_{\mathbf{K}} = (c - Hx)_{\mathbf{K}}, \qquad z_{\mathbf{K}}^T \Delta x_{\mathbf{K}} = 0,$$

where $\eta \in \mathbb{R}$ is a Lagrange multiplier. Let

$$[u, v] := H_{\mathbf{K},\mathbf{K}}^{-1}[(c - Hx)_{\mathbf{K}}, z_{\mathbf{K}}]$$

be evaluated by using a Cholesky factorization of $H_{\mathbf{K},\mathbf{K}}$. By positive definiteness of $H_{\mathbf{K},\mathbf{K}}$, it follows $z_{\mathbf{K}}^T v > 0$ and $\eta := (z_{\mathbf{K}}^T u)/(z_{\mathbf{K}}^T v)$ is well defined. Set

$$\Delta x_{\mathbf{K}} := u - \eta v. \tag{2.8}$$

Then, $z_{\mathbf{K}}^T \Delta x_{\mathbf{K}} = z_{\mathbf{K}}^T u - \frac{z_{\mathbf{K}}^T u}{z_{\mathbf{K}}^T v} z_{\mathbf{K}}^T v = 0$ and

$$H_{\mathbf{K},\mathbf{K}}\Delta x_{\mathbf{K}} + \eta z_{\mathbf{K}} = H_{\mathbf{K},\mathbf{K}}(u - \eta v) + \eta z_{\mathbf{K}} = (c - Hx)_{\mathbf{K}},$$

i.e., (2.8) defines the Newton step.

The next iterate is defined by using the maximum step length ≤ 1 that maintains the bound constraints. If a new constraint becomes active at the next iterate, the sets **A**, **K** and the Cholesky factor are updated with $O(n^2)$ operations as outlined below.

Else, if no new constraint becomes active at the next iterate, the sweep terminated. In this case, $x_{\mathbf{K}}$ is a global minimizer of q with respect to indices in **K** and subject to the equality constraint $z_{\mathbf{K}}^T x_{\mathbf{K}} = 0$, i.e., $\exists \mu \in \mathbb{R} : \; g_{\mathbf{K}} = \mu z_{\mathbf{K}}$.

2.5. **Cholesky updates.** There are five different rank-one-updates for a Cholesky factor presented in [10, pp. 514-523]. A rank-one-update also is available in `cholupdate` in Matlab and this is used in the numerical examples below. Octave also offers a command `choldelete` performing an elimination of a row and column of the matrix to be factored. Following the outline in [11], it is briefly described next how the deletion of a row and column amounts to a rank-one-update as available in `cholupdate`.

If a Cholesky factorization is given,

$$LL^T = \begin{bmatrix} L_{1,1} & 0 & 0 \\ l_{1,2}^T & l_{2,2} & 0 \\ L_{3,1} & l_{3,2} & L_{3,3} \end{bmatrix} \begin{bmatrix} L_{1,1}^T & l_{1,2} & L_{3,1}^T \\ 0 & l_{2,2} & l_{3,2}^T \\ 0 & 0 & L_{3,3}^T \end{bmatrix} = \begin{bmatrix} A_{1,1} & a_{1,2} & A_{3,1,}^T \\ a_{1,2}^T & a_{2,2} & a_{3,2}^T \\ A_{3,1} & a_{3,2} & A_{3,3} \end{bmatrix}$$

with lower case letters indicating column vectors and upper case letters indicating (sub-) matrices of appropriate dimensions, and if the row $(a_{1,2}^T, a_{2,2}, a_{3,2}^T)$ and the associated column are to be deleted such that

$$\widehat{L}\widehat{L}^T = \begin{bmatrix} \widehat{L}_{1,1} & 0 \\ \widehat{L}_{3,1} & \widehat{L}_{3,3} \end{bmatrix} \begin{bmatrix} \widehat{L}_{1,1}^T & \widehat{L}_{3,1}^T \\ 0 & \widehat{L}_{3,3}^T \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{3,1,}^T \\ A_{3,1} & A_{3,3} \end{bmatrix},$$

then $\widehat{L}_{1,1}\widehat{L}_{1,1}^T = A_{1,1} = L_{1,1}L_{1,1}^T$ implies $\widehat{L}_{1,1} = L_{1,1}$. From $\widehat{L}_{3,1}\widehat{L}_{1,1}^T = A_{3,1} = L_{3,1}L_{1,1}^T$, it follows $\widehat{L}_{3,1} = L_{3,1}$. Finally,

$$L_{3,1}L_{3,1}^T + l_{3,2}l_{3,2}^T + L_{3,3}L_{3,3}^T = A_{3,3} = \widehat{L}_{3,1}\widehat{L}_{3,1}^T + \widehat{L}_{3,3}\widehat{L}_{3,3}^T = L_{3,1}L_{3,1}^T + \widehat{L}_{3,3}\widehat{L}_{3,3}^T$$

implies $\widehat{L}_{3,3}\widehat{L}_{3,3}^T = L_{3,3}L_{3,3}^T + l_{3,2}l_{3,2}^T$, a rank-one-update of $L_{3,3}$. This update is stable as the term $l_{3,2}l_{3,2}^T$ is added to the existing Cholesky factor. (Augmenting a given factorization by a row and a column is numerically less stable, in general.)

2.6. **Global convergence.** Global convergence of the algorithm follows immediately from the derivations of the previous sections using a standard argument of active set methods: At the end of each sweep, the minimizer of q on the active set is generated. Since each step of the algorithm is a ("strict") descent step, this active set will never be revisited again. As there are only finitely many different active sets, the algorithm will terminate after a finite number of outer iterations, each of which taking at most n steps in the up-cycle and at most n steps during the sweep following thereafter.

2.7. **Technical details.** When performing a line search along a Newton direction, it might happen that two inactive indices turn active at the same time. For simplicity only one index is added to the active set. In the next Newton step, the step length might be zero and the set of active indices will then be increased again using another update of the Hessian matrix.

For Gaussian kernels with small exponents, the matrix H may be very ill-conditioned and the norm of the optimal solution of (1.1) may be huge. In such situations, interior-point methods generally are unstable. Large active upper bounds such as $C = 10^{10}$ may lead to a rather high numerical error in the optimal solution. (In fact, this was the reason why the work on the present paper using active sets was started.) For the numerical experiments in Section 4, a small multiple of the identity was added (as a regularization term) to H before forming the Cholesky factor. The result of the regularized Newton step then was corrected using one or two steps of iterative refinement (where the residuals for the iterative refinement corrections were computed with the original matrix H.) When the norm of the optimal solution often is huge the computation of $g_{\mathbf{K}} = (Hx - c)_{\mathbf{K}}$ frequently is subject to large cancellation errors. To take this into account, a relative KKT condition is listed in the numerical section, using $\|\widetilde{g}_{\mathbf{K}}\|_2/\|x\|_\infty$ in place of $\|\widetilde{g}_{\mathbf{K}}\|_2$. In such situations also the evaluation of $q(x)$ may be subject to high cancellation errors.

As an obvious technical detail, in the numerical implementation the definition (1.2) of p is changed using an ε-tolerance: more precisely, the cases $x_i \le \varepsilon$, $x_i \in (\varepsilon, (1-\varepsilon)C)$ and $x_i \ge (1-\varepsilon)C$ are distinguished where $1 \gg \varepsilon > 0$ is a user-defined tolerance for the active set. Likewise, definition (1.3) is modified $\mathbf{A} := \{i \mid x_i \le \varepsilon\} \cup \{i \mid x_i \ge (1-\varepsilon)C\}$, and the same for the definition (1.4).

We close the discussion of technical details with the remark that for small positive values of C, a step in the up-cycle might lead to a step length that makes another inactive variable active, and thus leads to repeated short steps in the up-cycle, oscillating between lower and upper bound. Since typical values of C for SVM are rather large, we did not concentrate on this case but just stopped the up-cycle after at most n steps. Alternatively, one could shorten the step length in the up-cycle to ensure that no inactive variable becomes active.

3. A Greedy SMO Algorithm

The SMO algorithm [1] consists of repeating the choice of $s = \pm e_i \pm e_j$ for properly selected indices $i \ne j$ as in Step 2 of the up-cycle, followed by a line search $x \leftarrow x + \lambda s$. The rules used in the up-cycle of CMU, however, differ from the rules for SMO since CMU selects active indices i, j whenever possible.

It is well known (see [12] for example) that alternating direction methods such as SMO may fail to converge if the selection of the directions is not carried out carefully, an aspect that is addressed later in this section. The original "SMO-paper" [1] refers to a general class of algorithms to establish global convergence. In several subsequent papers (see, for example, [13] and the references therein), numerous variants of the selection of i and j in the SMO algorithm have been proposed and their convergence has been established.

For the numerical experiments in this paper a greedy selection of the indices i, j in the SMO algorithm as well as a randomized selection are compared. The computational cost of the greedy selection is of order n operations, i.e. of the same order as the cost for one SMO step with any other pivoting rule, and it is substantially cheaper than one step of the sweeping cycle of CMU.

Before addressing the shortfalls of the greedy selection, it is outlined next:

At each step of the greedy SMO, the projected gradient $\widetilde{g}$ (see (2.1)) is used and updated. Let a feasible iterate x be given and set $g := Hx - c$ and $\widetilde{g} := \Pi_{\mathcal{N}} g = g - \mu z$ with $\mu = g^T z/n$. (This choice of μ differs from Proposition 1.1.)

A sparse (numerically cheap) search direction s starting at a feasible iterate x of (1.1) is defined using two indices i and j and setting $\widetilde{s} := \Pi_{\mathcal{B},x}(-\widetilde{g})$ as in (2.1). Then, $s := \operatorname{sign}(\widetilde{s}_i)(e_i - z_i z_j e_j)$. As in (2.6), it follows that $z^T s = 0$ and

$$q(x+\lambda s) = q(x) + \lambda \operatorname{sign}(\widetilde{s}_i)(g_i - z_i z_j g_j) + \tfrac{\lambda^2}{2}(H_{i,i} + H_{j,j} - 2 z_i z_j H_{i,j}).$$

Using (2.2), $z^T s = 0$ implies $(g_i - z_i z_j g_j) = g^T s = \widetilde{g}^T s = (\widetilde{g}_i - z_i z_j \widetilde{g}_j)$ and

$$q(x+\lambda s) = q(x) + \lambda \operatorname{sign}(\widetilde{s}_i)(\widetilde{g}_i - z_i z_j \widetilde{g}_j) + \tfrac{\lambda^2}{2}(H_{i,i} + H_{j,j} - 2 z_i z_j H_{i,j}), \tag{3.1}$$

where $\|\widetilde{g}\|_2 \le \|g\|_2$. Based on (3.1), the indices i and j are chosen successively:

(1) First, a greedy approach selects i as the entry maximizing $|\widetilde{s}_i|$. Then, set

$$\overline{\lambda}_i := \begin{cases} -x_i & \text{if } \widetilde{s}_i \le 0 \\ C - x_i & \text{if } \widetilde{s}_i > 0. \end{cases}$$

(2) Given i and the associated value $\overline{\lambda}_i$, again a greedy heuristics is used to define j such that q in (3.1) is minimized along $s = \operatorname{sign}(\widetilde{s}_i)(e_i - z_i z_j e_j)$ subject to box constraints. More precisely, for $j \ne i$ with $\widetilde{g}_i(\widetilde{g}_i - z_i z_j \widetilde{g}_j) > 0$, let[1]

$$\hat{\lambda}_j := -(\widetilde{g}_i - z_i z_j \widetilde{g}_j)/(H_{i,i} + H_{j,j} - 2 z_i z_j H_{i,j}).$$

Then $\overline{\lambda}_j$ is determined as

$$\overline{\lambda}_j := \begin{cases} \max\{\hat{\lambda}_j, \ -x_j\} & \text{if } \hat{\lambda}_j \le 0 \\ \min\{\hat{\lambda}_j, C - x_j\} & \text{if } \hat{\lambda}_j > 0. \end{cases}$$

Finally, set $\overline{\lambda}_j := \operatorname{sign}(\overline{\lambda}_j)\min\{|\overline{\lambda}_i|, |\overline{\lambda}_j|\}$. Then j is selected as to minimize

$$\overline{\lambda}_j(\widetilde{g}_i - z_i z_j \widetilde{g}_j) + \frac{\overline{\lambda}_j^2}{2}(H_{i,i} + H_{j,j} - 2 z_i z_j H_{i,j}).$$

Remark 3.1. For Gaussian kernels satisfying $H_{i,i} \equiv 1$ for all i, the selection of j can be carried out with about $20n$ floating point operations. For large values of n this effort may pay off by the reduction of the objective value. Choosing i and j simultaneously might result in an even larger

[1]The restriction $\widetilde{g}_i(\widetilde{g}_i - z_i z_j \widetilde{g}_j) > 0$ implies that $\operatorname{sign}(\hat{\lambda}_j) = \operatorname{sign}(\overline{\lambda}_i)$ $(= \operatorname{sign}(\widetilde{s}_i) = -\operatorname{sign}(\widetilde{g}_i))$.

reduction of q but generally, this would require order n^2 operations considering all entries $H_{i,j}$ for $1 \le i \le j \le n$.

Remark 3.2. The heuristics of choosing i can result in a choice of i with x_i close to the boundary of $\mathscr{B}$ and such that only a very short step length is possible, no matter how j is chosen. In this case, however, i will become active in the next step and $\tilde{s}_i$ will then be zero. For several other seemingly profitable choices of i one can construct examples that may lead to convergence to non-optimal points.

When i, j have been selected, the line search along $s = \text{sign}(\widetilde{s_i})(e_i - z_i z_j e_j)$ leads to the step length $\overline{\lambda}_j$ (possibly negative) that was computed during the selection process of i, j. Then g is updated as $g \leftarrow g + \overline{\lambda}_j(He_i - Hz_i z_j e_j)$ with $3n$ floating point operations. Thereafter $\widetilde{g}$ is updated as $\widetilde{g} \leftarrow g - \frac{g^T z}{n} z$ with another $3n$ operations. The greedy SMO algorithm stops when (1.5), (1.7), (1.8) are satisfied up to some tolerance ε or when a given maximum number of iterations has been reached. Choosing i and j uniformly randomly from $\{1, \dots, n\}$ is about 10 times cheaper than the above greedy heuristic, and it is guaranteed to converge without the danger of running into a cycle. However, in the numerical examples, the random choice is more than 10 times slower than the greedy heuristic; the latter one therefore is used for a conceptual comparison.

4. Numerical Experiments

Some numerical experiments were carried out to compare the overall solution times and the final accuracy of the solutions generated by CMU. The experiments were carried out using Gaussian kernels. As argued in [2], the Gaussian kernels are optimal with respect to a self-concordance parameter similar to the one introduced for barrier functions in [14].

Two sets of examples were used, half-moon shapes and checker board patterns. The solution times are always listed in seconds.

4.1. Half-moon shapes.

Half-moon shapes. The first set of examples uses a higher-dimensional half-moon shape, i.e., a connected non-convex set in $d \ge 2$ dimensions. The input for this example is: $d \ge 2$ (dimension of the data space), $\delta \in (0, 2)$, and n (number of training points).

In the numerical experiments, unless stated otherwise, the parameters $\delta = \frac{1}{4}$ and $n = 500$ are used. First, one defines a set $S_1 = \{x \in \mathbb{R}^d \mid \|x\|_2 \le 1, \ \|x + \delta e_1\|_2 \ge 1\}$. For $\delta = \frac{1}{4}$ and for any $i \in \{2, \dots, d\}$, the projection of S_1 onto the (x_1, x_i)-plane has a half-moon shape. In general dimensions $d \ge 2$, the set S_1 is the difference of two Euclidean balls that have non-empty intersection.

The SVM then is to decide whether a given "new" data point lies in S_1 or not. By definition, $S_1 \subset [-1, 1]^d$ but for large d the volumes are $\text{vol}(S_1) \ll \text{vol}([-1, 1]^d) = 2^d$ even when δ is large. In this case, drawing the training points uniformly from $[-1, 1]^d$ would result in extremely unbalanced labels. Likewise for the test points. Therefore, one defines $S^- = S_1 - \delta e_1$. For $x \in S^-$, it then follows $x + \delta e_1 \in S_1$, i.e., $\|x + \delta e_1\|_2 \le 1$, so that the interiors of S_1 and S^- are disjoint. Similarly, let $S^+ = S_1 + \delta e_1$; and then also the interior of S^+ is disjoint from S_1. Then, set $S_2 = S^- \cup S^+$.

The sample points $x = x^i$ are then generated with some random distribution[2] within the union of S_1 and S_2 and with labels $z_i = 1$ if $x \in S_1$ and $z_i = -1$ if $x \in S_2$. (The definition of S_2 implies that the SVM is to generate a two-sided approximation of S_1.)

For the numerical examples below, 10^5 test points drawn from the same distribution were always used to assess the classification error.

4.1.1. *Parameter selection.* For simplicity, all training points were classified correctly and the soft margin constant C therefore was set to $C = \infty$. First a comparison of the constants γ in the Gaussian kernel was carried out for $d = 2$ dimensions of the data space. (The kernel is based on the function $e^{\gamma\|x-y\|_2^2}$.) The results are shown in Table 1.

γ	cycles	iterations	time	KKT violation	$q(x^{final})$	$\|x^{final}\|_\infty$	rel class. errors
0.03	3	433	0.69	1.8e-11	-1.0e+12	5.8e+10	0.0106, 0.0267
0.3	7	1328	1.21	4.3e-16	-3.4e+09	2.9e+09	0.0327, 0.0223
3	6	832	0.85	5.1e-16	-4.1e+06	2.0e+06	0.0359, 0.0129

Table 1. (CMU classification errors depending on γ for $d = 2$.)

The number of cycles is the same as the number of Cholesky factorizations that were computed. The number of iterations is the number of steps in up- or down- cycles where each iteration used at most "order n^2 operations", somewhat less when the size of the inactive set was small. (Since $n = 500$ these examples used less than $3n$ iterations.) The time is on a ThinkPad from 2016, Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz. The KKT violation is the relative violation as detailed in Section 2.7. The relative classification error is the relative number of points which should have been classified in S_1 but were not, and likewise for S_2.

4.1.2. *Comparison with SMO.* In Section 4.1.1, the overall classification errors for $\gamma = 0.03$ are slightly lower than for the other values. When choosing even smaller values of $\gamma = 0.03$ for this example, the Hessian of H is so poorly conditioned that a reliable solution of (1.1) with the standard numerical precision of about 16 decimal digits was not possible with any of the methods. For the value $\gamma = 0.03$ the performance of CMU was compared to SMO with greedy selection of the search direction (GSMO) and with the much cheaper random selection (RSMO) in the next table. To compensate for the cheaper iterations, in GSMO the maximum number of iterations was set to $1000n$ while it was set to $10000n$ for RSMO (both, in Table 2 and Table 3).

method	time	KKT violation	$q(x^{final})$	rel class. errors
CMU	0.69	1.8e-11	-1.0e+12	0.0106, 0.0267
GSMO	17.9	2.0e-08	-1.5e+10	0.3377, 0.3537
RSMO	22.0	1.8e-07	-1.7e+09	0.3649, 0.3051

Table 2. (Different algorithms for $\gamma = 0.03$ and $d = 2$.)

The Hessian matrix for this problem is very ill-conditioned. In spite of the fact that a large number of iterations was allowed leading to a solution of reasonably high accuracy, the classification error of GSMO or RSMO for this problem was considerably higher than for CMU. (The

[2]More precisely, a point x in S_1 is generated as follows: First x_1 is generated uniformly from $[-\delta/2, 1]$. Then, a random normal vector $y \in \mathbb{R}^{d-1}$ is drawn and $\bar{y} := y/\|y\|_2$ is defined. Then, $x := [x_1; \lambda y]$, where λ is drawn uniformly from $[\delta_1, \delta_2]$ with $\delta_1 = \sqrt{\max\{0, 1-(x_1+\delta)^2\}}$, $\delta_2 = \sqrt{1-x_1^2}$. Half of the training points are then shifted to both parts of the set S_2. The training points generated this way are concentrated at both ends of the "half moon" and fewer points in the middle, an effect that is even more pronounced for $d > 2$.

minimum value of q must be less or equal to the value returned by CMU, indicating that also the values of q generated by GSMO or RSMO are far from optimality.)

The classification error is used, for example, in cross validation approaches for the parameters of the kernel, and thus it is important that the numerical accuracy is sufficiently high not to deteriorate the classification error. In the artificial example above, the accuracy generated by either variant of SMO was not sufficient. There are many other versions of SMO with different choices of the pivot element, but they all share the property that high accuracy solutions require very many steps of SMO.

With a slightly different setting of $d \in \{3,5\}$ dimensions, a much larger number of $n =$ 10000 training data points was compared on a small computer cluster with 8 sockets, 64 CPUs, AMD Opteron(tm) Processor 6282 SE. The dimension of H was 10000 by 10000, and for such dimensions, the computation times depend more closely on the numerical effort and to a lesser extent on the overhead caused by the fact that the Matlab program used is based on an (uncompiled) interpreter. (The overhead grows about linearly with the dimension and the computational effort grows more than quadratically.) The results are given in Table 3.

method	d	γ	time	KKT violation	$\|x^{final}\|_\infty$	$q(x^{final})$	rel class. errors
CMU	3	0.03	24491	3.5e-10	5.0e+10	-8.6e+12	0.0035, 0.0170
GSMO	3	0.03	18698	4.9e-09	4.9e+10	-3.2e+11	0.2565, 0.2619
RSMO	3	0.03	11610	1.5e-06	5.7e+06	-3.8e+09	0.1818, 0.2921
CMU	5	0.03	24187	3.6e-10	4.7e+10	-8.4e+12	0.0068, 0.0223
GSMO	5	0.03	19195	4.1e-07	2.2e+08	-1.9e+10	0.4113, 0.4458
RSMO	5	0.03	11633	2.0e-06	6.7e+06	-2.5e+09	0.1673, 0.3509
CMU	5	3	27156	1.2e-14	2.4e+05	-9.1e+05	0.0679, 0.0789
GSMO	5	3	5959	8.5e-11	2.4e+05	-9.1e+05	0.0671, 0.0774
RSMO	5	3	11656	3.4e-04	1.8e+04	-5.7e+05	0.0541, 0.0833

Table 3. (Different algorithms for $\gamma \in \{0.03, 3\}$ and $n = 10000$.)

Again, these numbers indicate that a small value of γ should at least be considered in a cross validation approach, and that for small values of γ a high numerical accuracy is required, in order to reduce the classification error.

The number of outer iterations in CMU in these examples was between 6 and 8 with 11460 to 14054 inner iterations while GSMO and RSMO used $1000n$ and $1000n$ iterations respectively. Only for $\gamma = 3$, GSMO terminated early after $350n$ iterations because the stopping criterion – which was always tested after integer multiples of n iterations – was satisfied.

For CMU, the size of the first set of inactive indices was limited to 2000 in order to start with a moderately cheap Cholesky factorization, possibly at the expense of more cycles. (This limitation does not apply to Table 2, where $n = 500$.)

Comparing GSMO and RSMO, it is remarkable, that a more elaborate choice of pivots leads to substantially higher numerical accuracy (i.e., lower values of q) even when considering the numerical effort and allowing for 10 times more random steps.

4.1.3. *Dependence on d.* The next example illustrates the impact of the dimension d on the performance of CMU with $n = 500$ training points.

d	cycles	iterations	time	KKT violation	$q(x^{final})$	$\|x^{final}\|_\infty$	rel class. errors
2	3	433	0.69	1.8e-11	-1.0e+12	5.8e+10	0.0106, 0.0267
3	3	518	0.77	1.1e-11	-1.2e+12	8.4e+10	0.0250, 0.0669
5	7	686	1.05	2.4e-12	-1.5e+11	5.8e+10	0.1728, 0.2001
10	5	499	0.88	7.1e-15	-1.7e+09	7.6e+08	0.4260, 0.3555
50	3	260	0.60	9.8e-15	-1.6e+07	6.8e+06	0.4849, 0.3786

Table 4. (Classification errors depending on d for $\gamma = 0.03$.)

The condition numbers of $H \in \mathbb{R}^{n\times n}$ improve for larger values of d but the classification errors deteriorate. (For $d > 3$ more than $n = 500$ points might be necessary to generate a classifier with less than 10% relative classification error.) In any case, the purpose of Table 4 was to illustrate the effect of higher dimensions of the data space on the running times and the numerical accuracy of CMU.

4.2. **Checker board pattern.** In this example, the sets S_1 and S_2 were defined along a 3×3 checker board pattern, and again 500 (uniformly randomly defined) training points were used without errors in the classifications of the training set. Figure 1 shows the regions separated by the SVM with Gaussian kernel with two different values of γ. The larger value of γ results in "more curvature" of the boundary of the classification sets (in green and red).

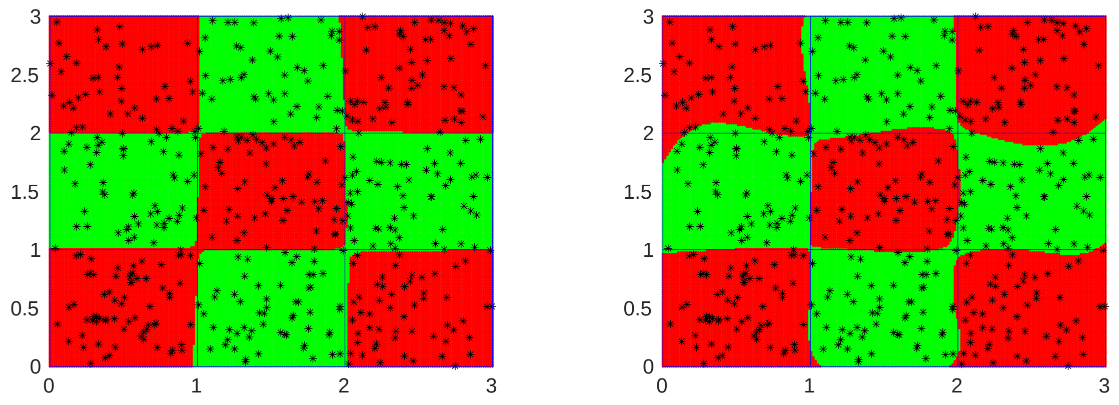

Figure 1, CMU for Gaussian kernel with $C = \infty$ and $\gamma = 0.03$ and $\gamma = 3$ from left to right.

Figure 1 suggests that the curvature of the boundary is lower for smaller values of c, a fact that has been analyzed theoretically in [2] based on a modified self-concordance property of [14].

We point out that also in the plot on the right all training points are classified correctly, and in fact, if no further information is given, it might well be that the pattern on the right describes the "true pattern". For the "exact checker board pattern" that was actually used, the above plot gives a visual indication that in some cases, small values of the parameter γ may be appropriate.

On the one side, as argued in [2], small values of γ imply small values of a certain self-concordance parameter similar to the one introduced in [14], on the other side, small values of γ result in a very ill-conditioned matrix H. For $\gamma = 0.03$ there were some steps (less than 1 percent of the Newton steps), where the Newton step resulted in a numerical increase of the value of q. (We tested some of these instances: even though the Newton step s satisfied $\nabla q(x)^T s <$

$-10^{-5}\|\nabla q(x)\|_2\|s\|_2$ the computed values of q did satisfy $q(x+\frac{1}{10}s) > q(x)$.) Therefore smaller values of γ are not included in the numerical examples of this section.

The same pattern with $\gamma = 0.03$ is now used with GSMO with $1000n$ and $10000n$ iterations. (Here, as well, RSMO generated less accurate results.)

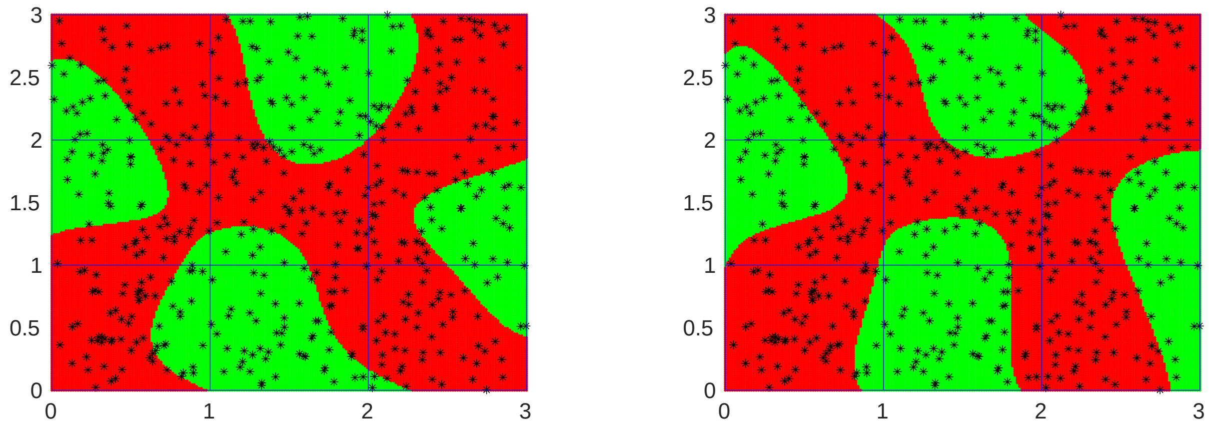

Figure 2, Same example, illustrating that high numerical accuracy is essential. GSMO for Gaussian kernel with $\gamma = 0.03$, $1000n$ and $10000n$ iterations from left to right.

Even when $10000n = 10^8$ iterations are allowed, the accuracy of the solution is rather low, and there are many falsely classified training data points. Some of the results of above test runs are listed below. While the tendency observed in Section 4.1 continues here as well these examples are not intended to make a general claim about the efficiency but to point out that there are instances for which the computation of a high accuracy solution with moderate computational effort may be essential.

method	iterations	time	KKT violation	$q(x^{final})$	$\|x^{final}\|_\infty$
CMU	588	0.99	2.2e-11	-1.4e+11	4.6e+10
GSMO	500000	18.25	5.3e-08	-6.2e+08	1.1e+08
GSMO	5000000	197.7	9.5e-09	-3.8e+09	5.7e+08

Table 5. (Checker board example, convergence of CMU and GSMO for $\gamma = 0.03$ and $n = 500$.)

All files used to generate above data are available at

https://github.com/florianjarre/SVM-Test-Set

5. Conclusion

A key observation used in the CMU algorithm lies in the fact that repeated stable and numerically cheap increases of the inactive set are possible while reducing the objective function and without relying on a Hessian factorization. In the preliminary numerical experiments, only a small number of cycles were needed so that the overall numerical effort was small. The implementation in [15] includes a simple iterative refinement step that helps reducing the numerical rounding errors.

Acknowledgments

The author likes to thank Kevin Wischnewski for helpful comments correcting and improving the presentation of this paper.

References

[1] J. Platt, Sequential minimal optimization: A fast algorithm for training support vector machines. MSR-TR-98-14, Microsoft, https://www.microsoft.com/en-us/research/publication/sequential-minimal-optimization-a-fast-algorithm-for-training-support-vector-machines

[2] F. Jarre, A mathematical introduction to SVMs with self-concordant kernel, V.L. Turova, A.E. Kovtanyuk and J. Zimmer (eds.), MMSC 2024, EPiC Series in Computing, vol. 104, 126-150, 2024. https://easychair.org/publications/paper/2msG/open

[3] V.N. Vapnik, A.Y. Chervonenkis, The necessary and suffcient conditions for consistency in the empirical risk minimization method, Pattern Recognition and Image Analysis 1 (1991), 283-305.

[4] B. Schölkopf, A.J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. (Adaptive Computation and Machine Learning Series), 2001.

[5] T. Hofmann, B. Schölkopf, A.J. Smola, Kernel methods in machine learning, Ann. Statist. 36 (2008), 1171-1220

[6] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, A comprehensive survey on support vector machine classification: Applications, challenges and trends. Neurocomputing 408 (2020), 189-215.

[7] M.C. Ferris, T.S. Munson, Interior-point methods for massive suppport vector machines, SIAM J. Optim. 13 (2003), 783-804.

[8] D. Ma, M. Saunders, SMO vs PDCO for SVM: Sequential Minimal Optimization vs Primal-Dual interior method for Convex Objectives for Support Vector Machines. Working paper, Dept of Management Science and Engineering, Stanford University, https://web.stanford.edu/group/SOL/reports/Ma-SMOvsPDCOforSVM.pdf, 2015.

[9] T.A. Davis, W.A. Hager, Modifying a sparse cholesky factorization, SIAM J. Matrix Anal. Appl. 20 (1999), 606–627.

[10] P.E. Gill, G.H. Golub, W. Murray, M.A. Saunders, Methods for modifying matrix factorizations, Math. Comput. 28 (1974), 505–535.

[11] P. Patrinos, P. Sopasakis, H. Sarimveis, A global piecewise smooth Newton method for fast large-scale model predictive control, Automatica, 47 (2011), 2016-2022,

[12] M.J.D. Powell, Direct search algorithms for optimization calculations, Acta Numer. 7 (1998), 287-336.

[13] S. Keerthi, E. Gilbert, Convergence of a generalized SMO algorithm for SVM classifier design, Machine Learning 46 (2002), 351-360.

[14] Y. Nesterov, A. Nemirovskii, Interior-point polynomial algorithms in convex programming, SIAM, Philadelphia, 1994.

[15] F. Jarre, SVM-Test-Set in Matlab, https://github.com/florianjarre/SVM-Test-Set, 2025.