Journal of Machine Learning Research 12 (2011) 141-202 Submitted 3/09; Revised 7/10; Published 1/11

Training SVMs Without Offset

Ingo Steinwart INGO.STEINWART@MATHEMATIK.UNI-STUTTGART.DE
Institut für Stochastik und Anwendungen
Fakultät für Mathematik und Physik
Universität Stuttgart
Pfaffenwaldring 57
D-70569 Stuttgart, Germany

Don Hush DHUSH@LANL.GOV
ISR-2, Mail Stop B244
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

Clint Scovel JCS@LANL.GOV
CCS-3, Mail Stop B265
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

Editor: Sathiya Keerthi

Abstract

We develop, analyze, and test a training algorithm for support vector machine classifiers without offset. Key features of this algorithm are a new, statistically motivated stopping criterion, new warm start options, and a set of inexpensive working set selection strategies that significantly reduce the number of iterations. For these working set strategies, we establish convergence rates that, not surprisingly, coincide with the best known rates for SVMs with offset. We further conduct various experiments that investigate both the run time behavior and the performed iterations of the new training algorithm. It turns out, that the new algorithm needs significantly less iterations and also runs substantially faster than standard training algorithms for SVMs with offset.

Keywords: support vector machines, decomposition algorithms

1. Introduction

Historically, support vector machines (SVMs) were motivated by a geometrical illustration of their linear decision surface in the feature space. This illustration justified the use of an offset b that moves the decision surface from the origin. However, in recent years it has become increasingly evident that this geometrical interpretation has serious flaws, see, for example, Steinwart (2003) for some illustrations, when considering kernels that have a large feature space such as the Gaussian RBF kernels. In addition, the current approach, see, for example, Steinwart and Christmann (2008), for investigating the generalization performance of SVMs for classification does not suggest that the offset offers any improvement for such kernels. On the other hand, the SVM optimization problem with offset imposes more restrictions on solvers than the optimization problem without offset does. For example, the offset leads to an additional equality constraint in the dual optimization problem, which in turn makes it necessary to update at least two dual variables at each iteration of commonly used solvers such as sequential minimal optimization (SMO). In addition, such solvers can only update certain pairs of dual variables, namely the pairs whose update still satisfies the equality constraint. Moreover, the offset makes it relatively expensive to calculate the duality gap, see Cristianini and Shawe-Taylor (2000), which may serve as a stopping criterion for these solvers, and hence one usually considers upper bounds of this gap such as the one from the maximal violating pair algorithm, see, for example, Lin (2002b).

Despite these issues, research on algorithmic solutions has, with a few exceptions such as Kecman et al. (2005), Vogt (2002) and Huang et al. (2006), so far mostly focused on SVM formulations with offset. We refer to Lin (2001), Keerthi et al. (2001), Lin (2002a), Hush and Scovel (2003), List and Simon (2004), Fan et al. (2005), List and Simon (2005), Chen et al. (2006), Hush et al. (2006), Glasmachers and Igel (2006), List et al. (2007), List and Simon (2007) and the references therein. One motivation for this focus may be the fact that certain other SVM formulations such as one-class SVMs and SVMs for finding the smallest ball enclosing all data points do have an offset, and hence these formulations can be dealt with (almost) simultaneously. Moreover, it was noted early on that for SVMs with offset, the resulting equality constraint in the dual optimization problem can be avoided, if the offset is also penalized in the regularizer. The package BSVM by Hsu and Lin (2002) and Hsu and Lin (2006) implements this idea for the hinge loss, while Mangasarian and Musicant (2001) and Fung and Mangasarian (2001) use this idea in conjunction with other margin-based loss functions.

The goal of this work is to fill the described gap by developing algorithms for SVMs without offset. As it turns out, these algorithms not only achieve a classification accuracy that is comparable to the one for SVMs with offset, but also run significantly faster. This improvement is made possible by a couple of new algorithmic ideas that are not straightforward to implement for SVMs with offset. Inspired by recent results on the statistical performance of SVMs, see (Steinwart and Christmann, 2008, Chapter 7.4), the first idea is a new stopping criterion, which is, roughly speaking, a clipped duality gap. The second idea is a new working set selection strategy. As mentioned above, SMO type approaches for SVMs without offset can, in principle, update a single dual variable at each iteration. However, our experiments show that this approach does not lead to sufficiently fast training algorithms, and hence we will describe in detail, how an SMO type approach for two dual variables works. Of course, such an approach requires a good working set selection strategy. To identify one, we describe and test various strategies that try to find a pair of dual variables whose update approximately maximizes the gain in the dual objective function. Basically all these strategies first identify *one* dual variable whose update maximizes the gain in the dual objective and then search for a second variable that matches well to the first variable. Clearly, the first search is $O(n)$, where n is the number of samples, while the order for the second search will be between $O(1)$ and $O(n)$ depending on the particular strategy. Interestingly, we will see that certain *combinations* of $O(1)$ strategies for finding the second variable need almost as few iterations as an $O(n^2)$ search over *all* pairs. In particular, these combinations essentially need the same number of iterations as some natural $O(n)$ strategies for choosing the second dual variable do. Since each iteration of the latter strategies is obviously more expensive, the $O(1)$ combinations enjoy significantly shorter run times as will be seen in the experiments.

For solvers using the new stopping criterion and (combinations of) the working set strategies mentioned above, we further establish theoretical guarantees on the number of iterations performed. Not surprisingly, it turns out that the analysis without bias is less complicated than the one for the offset case, while the resulting guarantees coincide with the best known guarantees for solvers with offset. Recall that the latter can be obtained by combining the analysis of so-called rate certifying algorithms, see List and Simon (2005), Hush et al. (2006) and List and Simon (2007), with a recent analysis of the duality gap, see List et al. (2007). Unlike the rate certifying algorithms for SVMs with offset, however, our algorithms not only possess these guarantees, but also run significantly faster than typically implemented training algorithms, as our experimental section shows.

We also consider the possibility to initialize the solver with (transformed) previous solutions when working on a grid of hyper-parameters. Here it first turns out that the missing equality constraint gives us more freedom to transform these solutions. We describe and test several such transformations ranging from relatively simple to quite complex procedures. In the experiments, we then see that SVMs without offset profit more from simple warm start initializations than SVMs with offset do. In addition, the more complex warm start strategies, which cannot be directly implemented for SVMs with offset, lead to further improvements. In particular, for data sets containing a few thousand samples, SVMs without offset profit about twice as much from a good warm start strategy than SVMs with offset do. As a result, our SVMs without offset are approximately 7 times faster than SVMs with offset on these data sets, if the hyper-parameters are determined by a cross-validation approach.

This work is organized as follows: Section 2 introduces an SMO type algorithm for SVMs without offset that performs one dual variable update per iteration. We further describe the new stopping criterion based on a clipped duality gap as well as several warm start strategies. Section 3 then generalizes this algorithm to handle two variables at each iteration. In particular, we describe how to solve the corresponding two dimensional optimization problem exactly. Furthermore, we present several working set selection strategies. Section 4 contains our theoretical analysis, while the experiments can be found in Section 5. Finally, concluding remarks can be found in Section 6 and an appendix contains detailed data from our experiments.

2. The Basic Algorithm: Optimizing One Coordinate

Throughout this paper, we write $[t]_a^b := \max\{a, \min\{b,t\}\}$, $t \in \mathbb{R}$, $b > a$, for the clipping operation that clips a real number t whenever it is outside the interval $[a,b]$. To introduce SVMs without offset term, let us consider a training set $T = ((x_1,y_1),\dots,(x_n,y_n)) \in (X \times \{-1,1\})^n$ and a function $f : X \to \mathbb{R}$. Then the empirical hinge risk of f is defined by

$$\mathcal{R}_{L,T}(f) := \frac{1}{n}\sum_{i=1}^{n} w_i L(y_i, f(x_i))\,,$$

where L denotes the hinge loss $L(y,t) := \max\{0, 1-yt\}$, and $w_i > 0$ is a weight associated to the sample (x_i,y_i). For example, in ordinary binary classification we have $w_i = 1$ for all $i = 1,\dots,n$, whereas in weighted binary classification we have two real numbers $w_{\text{pos}} > 0$ and $w_{\text{neg}} > 0$ such that $w_i = w_{\text{pos}}$ if $y_i = 1$ and $w_i = w_{\text{neg}}$ if $y_i = -1$. In the following, we will exclusively consider the case of weighted binary classification, which, of course, includes the case of ordinary binary classification. Now the SVM without offset solves the problem

$$f_{T,\lambda} \in \underset{f \in H}{\arg\min}\, \lambda \|f\|_H^2 + \mathcal{R}_{L,T}(f)\,, \tag{1}$$

where H is the reproducing kernel Hilbert space (RKHS) of a kernel k. The statistical analysis of SVMs shows, see (Steinwart and Christmann, 2008, Corollary 5.34), that a necessary condition for learning in the sense of universal consistency is the strict positive definiteness of the kernel k. In the following, we adopt this point of view, partially also because for kernels that fail to be strictly positive definite the offset may actually improve the learning performance, both theoretically and practically. In other words, we assume throughout this paper that the *Gram matrix* $(k(x_i,x_j))_{i,j=1}^n$ is strictly positive definite whenever the data points $x_1,\dots,x_n$ are mutually distinct.[1] Considering the case $n=1$, it is then easy to conclude that $k(x,x)>0$ for all $x\in X$, and hence we may and will additionally assume that k is *normalized*, that is, $k(x,x)=1$ for all $x\in X$. Although this assumption is not really necessary, it makes the description of the algorithm significantly simpler. In addition, it is satisfied by many popular kernels on $X=\mathbb{R}^d$ such as the Gaussian RBF kernel $k(x,x'):=\exp\bigl(-\sigma^2\|x-x'\|_2^2\bigr)$, and the Poisson kernel $k(x,x'):=\exp\bigl(-\sigma\|x-x'\|_2\bigr)$, where in both cases $\sigma>0$ is called the width parameter. Furthermore, note that for strictly positive definite and normalized kernels we have $|k(x,x')|=1$ if and only if $x=x'$. For the Gaussian and Poisson kernel, this characterization is, of course, trivial, and in the general case, it quickly follows when considering the case $n=2$.

To derive an algorithm that produces an approximate solution of (1) we first multiply the objective function in (1) by $\frac{1}{2\lambda}$ and introduce slack variables. This leads to the following optimization problem:

$$\begin{array}{lll} \arg\min\limits_{(f,\xi)} & P_C(f,\xi):=\dfrac{1}{2}\|f\|_H^2+\sum\limits_{i=1}^n C_i\xi_i & \\ \text{s.t.} & \xi_i\geq 0\,, & i=1,\dots,n, \\ & \xi_i\geq 1-y_if(x_i)\,, & i=1,\dots,n, \end{array} \tag{2}$$

where $C_i:=\frac{w_{\text{pos}}}{2\lambda n}$ if $y_i=1$ and $C_i:=\frac{w_{\text{neg}}}{2\lambda n}$ otherwise. Analogously to the offset case, see, for example, (Cristianini and Shawe-Taylor, 2000, p. 107f), one can then show that the dual of this problem is

$$\begin{array}{lll} \max\limits_{\alpha\in\mathbb{R}^n} & W(\alpha):=\langle e,\alpha\rangle-\dfrac{1}{2}\langle\alpha,K\alpha\rangle & \\ \text{s.t.} & 0\leq\alpha_i\leq C_i\,, & i=1,\dots,n\,, \end{array} \tag{3}$$

where $e:=(1,\dots,1)\in\mathbb{R}^n$ and K is the $n\times n$ matrix with entries $K_{i,j}:=y_iy_jk(x_i,x_j)$, $i,j=1,\dots,n$. In addition, the Karush-Kuhn-Tucker (KKT) conditions are

$$\begin{aligned} \bigl(y_if(x_i)+\xi_i-1\bigr)\alpha_i &= 0, \qquad i=1,\dots,n,\\ (C_i-\alpha_i)\xi_i &= 0, \qquad i=1,\dots,n, \end{aligned}$$

and a solution $\alpha^*\in[0,C]:=[0,C_1]\times\cdots\times[0,C_n]$ of (3) yields a solution (f^*,ξ^*) of (2) by setting

$$f^*:=\sum_{i=1}^n y_i\alpha_i^*k(x_i,\,\cdot\,)$$

and $\xi_i^*:=\max\{0,1-y_if^*(x_i)\}$, $i=1,\dots,n$. Obviously, (3) is identical to the standard dual SVM problem besides the missing equality constraint $\langle y,\alpha\rangle=0$. Now recall that this equality constraint makes it necessary to update at least two coordinate values at a time to ensure feasibility, while in (3) we can update *single* coordinates. Some ideas for such a single direction update will be recalled in the following subsections to provide the background for working sets of size two considered in Section 3.

[1] If we have samples with $x_i=x_j$ for some $i\neq j$, the Gram matrix of a strictly positive definite kernel k, is, of course, no longer strictly positive definite. The algorithmic consequences of this observation will be discussed in detail in Section 3. Here, we only note that our solver will need a strictly positive kernel, but not a strictly positive Gram matrix.

2.1 Working Sets of Size One

To recall the one-dimensional update step, see also (Cristianini and Shawe-Taylor, 2000, p. 131ff), we define

$$\nabla W_i(\alpha) := \frac{\partial W}{\partial \alpha_i}(\alpha) = 1 - \sum_{j=1}^{n} \alpha_j K_{i,j}\,.$$

Moreover, for an $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$ and an index $i \in \{1, \ldots, n\}$ we write $\alpha^{\setminus i} := \alpha - \alpha_i e_i$, where e_i denotes the i-th vector of the standard basis of $\mathbb{R}^n$, that is, $\alpha^{\setminus i}$ equals α in all coordinates except the i-th, where it equals zero. Now basic calculus together with $K_{i,i} = 1$ for normalized kernels shows that

$$\tilde{\alpha}_i \mapsto W(\alpha^{\setminus i} + \tilde{\alpha}_i e_i) = \langle \alpha^{\setminus i}, e\rangle + \tilde{\alpha}_i - \frac{1}{2}\langle \alpha^{\setminus i}, K\alpha^{\setminus i}\rangle - \tilde{\alpha}_i \langle e_i, K\alpha^{\setminus i}\rangle - \frac{1}{2}\tilde{\alpha}_i^2$$

attains its *global* maximum over $\mathbb{R}$ at

$$\alpha_i^* = 1 - \langle e_i, K\alpha^{\setminus i}\rangle = 1 - \sum_{j \neq i} \alpha_j K_{i,j} = \nabla W_i(\alpha) + \alpha_i\,.$$

Obviously, if $\alpha_i^* \in [0, C_i]$, the function $\alpha_i \mapsto W(\alpha^{\setminus i} + \alpha_i e_i)$ also attains its maximum over $[0, C_i]$ at α_i^*. On the other hand, if, for example, $\alpha_i^* > C_i$, then a simple concavity argument shows that the function attains its maximum over $[0, C_i]$ at C_i. By this and an analogous consideration in the case $\alpha_i^* < 0$ we hence see that the function $\alpha_i \mapsto W(\alpha^{\setminus i} + \alpha_i e_i)$ attains its maximum over $[0, C_i]$ at

$$\alpha_i^{new} := [\nabla W_i(\alpha) + \alpha_i]_0^{C_i}\,. \tag{4}$$

The next question is in which coordinate i should we perform the update. A simple and straightforward approach, see, for example, (Cristianini and Shawe-Taylor, 2000, p. 132), is to update for each coordinate $i = 1, \ldots, n$ iteratively. A more advanced idea, see Vogt (2002) and also (Huang et al., 2006, Chapter 3), is to choose KKT violators for the update, that is, indices that, for a specified $\varepsilon > 0$, satisfy

$$\begin{array}{rccl} & \alpha_i < C_i & \text{and} & \nabla W_i(\alpha) > \varepsilon\,, \\ \text{or} & \alpha_i > 0 & \text{and} & \nabla W_i(\alpha) < -\varepsilon\,. \end{array} \tag{5}$$

Obviously, the extreme case of this approach is to look for the indices

$$\begin{aligned} i^*_{\text{up}} &\in \arg\max\{\nabla W_i(\alpha) : \alpha_i < C_i\}\,, \\ i^*_{\text{down}} &\in \arg\min\{\nabla W_i(\alpha) : \alpha_i > 0\} \end{aligned}$$

and to pick the index of these two candidates whose gradient has the larger absolute value. Another idea, which is motivated by Glasmachers and Igel (2006), Hush et al. (2006), Hush and Scovel (2003) and List and Simon (2005), is to choose the coordinate i^* whose update achieves the largest improvement for the objective dual value $W(\alpha)$. In other words, it performs the update in the direction

$$i^* \in \arg\max_{i=1,\ldots,n} W(\alpha + \delta_i e_i) - W(\alpha)\,, \tag{6}$$

where $\delta_i := \alpha_i^{new} - \alpha_i$ denotes the difference between the new and the old value of α_i. Using the following trivial lemma, it is easy to see that Procedure 1 solves (6).

Lemma 1 *For $\delta \in \mathbb{R}$ and $i = 1,\dots,n$ we have*

$$W(\alpha+\delta e_i) - W(\alpha) = \delta \cdot (\nabla W_i(\alpha) - \delta/2)\,.$$

Proof By the symmetry of K we find

$$\langle \alpha, K\alpha\rangle - \langle \alpha+\delta e_i, K(\alpha+\delta e_i)\rangle = -2\delta\langle \alpha, Ke_i\rangle - \delta^2\,.$$

Combining this with $\langle e, \alpha+\delta e_i\rangle - \langle e,\alpha\rangle = \delta$ yields the assertion. ■

Procedure 1 Calculate $i^* \in \arg\max_{i=1,\dots,n} \delta_i \cdot (\nabla W_i(\alpha) - \delta_i/2)$

```
$bestgain \leftarrow -1$
for $i = 1$ to $n$ do
    $\alpha_i^* \leftarrow [\nabla W_i(\alpha) + \alpha_i]_0^{C_i}$
    $\delta \leftarrow \alpha_i^* - \alpha_i$
    $gain \leftarrow \delta \cdot (\nabla W_i(\alpha) - \delta/2)$
    if $gain > bestgain$ then
        $bestgain \leftarrow gain$
        $i^* \leftarrow i$
    end if
end for
```

2.2 Stopping Criteria

Several stopping criteria for the SVM *with* offset have been proposed and a straightforward approach is to adapt one of these. For example, one can stop if both $\nabla W_{i^*_{\text{up}}}(\alpha) \le \varepsilon$ and $\nabla W_{i^*_{\text{down}}}(\alpha) \ge -\varepsilon$, that is, if the KKT conditions are satisfied up to some predefined $\varepsilon > 0$. Another simple idea is to use the duality gap as a stopping criterion, see, for example, (Cristianini and Shawe-Taylor, 2000, p. 109 & 128). For SVMs *without* offset this duality gap is of the form

$$\text{gap}(\alpha) := \langle \alpha, K\alpha\rangle - \langle e,\alpha\rangle + \sum_{i=1}^{n} C_i\,[\nabla W_i(\alpha)]_0^\infty \;\le\; \varepsilon\,, \tag{7}$$

where $\varepsilon > 0$ does not necessarily have the same value as above.

In this work, however, we consider a little more involved stopping criterion that is based on recent results from the statistical analysis of SVMs in Steinwart et al. (2007). Namely, it was shown in Steinwart et al. (2007) that an $f^* \in H$ satisfying

$$\lambda\|f^*\|_H^2 + \mathcal{R}_{L,T}([f^*]_{-1}^1) \le \min_{f\in H} \lambda\|f\|_H^2 + \mathcal{R}_{L,T}(f) + \varepsilon \tag{8}$$

for yet another pre-defined $\varepsilon > 0$ satisfies the same oracle inequality up to 4ε as the true solution $f_{T,\lambda}$. Moreover, a more careful analysis of Steinwart et al. (2007) shows that the factor 4 can be essentially removed, so that for say $\varepsilon := 0.001$ the learning guarantees for the approximate solution f^* are at most 0.1% worse than those for the true solution $f_{T,\lambda}$. To develop a stopping criterion from this observation, we denote the minimum of the objective function P_C in (2) by P_C^*. Moreover, for a dual point $\alpha \in [0,C]$ we define, as usual, a corresponding primal function by

$$f := \sum_{i=1}^{n} \alpha_j y_j k(x_j, \cdot)$$

and its corresponding slack variables by $\xi_i := \max\{0, 1 - y_i f(x_i)\}$, $i = 1, \dots, n$. Using $1 - y_i f(x_i) = \nabla W_i(\alpha)$ and $\|f\|_H^2 = \langle \alpha, K\alpha \rangle$ as well as $P_C^* \geq W(\alpha) = \langle e, \alpha \rangle - \langle \alpha, K\alpha \rangle / 2$ and

$$\max\bigl\{0, 1 - y[t]_{-1}^{1}\bigr\} = 1 - y[t]_{-1}^{1} = [1 - yt]_0^2$$

for all $y = \pm 1$, $t \in \mathbb{R}$, we hence see that (8) is satisfied if

$$S(\alpha) := \langle \alpha, K\alpha \rangle - \langle e, \alpha \rangle + \sum_{i=1}^{n} C_i [\nabla W_i(\alpha)]_0^2 \ \leq\ \frac{\varepsilon}{2\lambda}\,. \tag{9}$$

Note that the statistical analysis of Steinwart et al. (2007) also suggests that the right hand side of (7) can be replaced by $\frac{\varepsilon}{2\lambda}$, where ε has the same value as in (9). Consequently, the difference between these two stopping criteria is the fact that (9) considers *clipped* slack variables, which may be substantially smaller than the unclipped slack variables used in (7). Moreover, unlike the duality gap stopping criterion for SVMs *with* offset, see (Cristianini and Shawe-Taylor, 2000, p. 109f), both (7) and (9) are directly computable since they do not require the offset.

To efficiently compute $S(\alpha)$ we first observe that the first two terms of the updated $S(\alpha + \delta e_i)$ can be easily computed from the first two terms of $S(\alpha)$. Indeed, if we write

$$\begin{aligned} T(\alpha) &:= \langle \alpha, K\alpha \rangle - \langle e, \alpha \rangle\,, \\ E(\alpha) &:= \sum_{i=1}^{n} C_i [\nabla W_i(\alpha)]_0^2\,, \end{aligned}$$

then we have $S(\alpha) = T(\alpha) + E(\alpha)$, and the calculations in the proof of Lemma 1 immediately show

$$T(\alpha + \delta e_i) = T(\alpha) - \delta\bigl(2\nabla W_i(\alpha) - 1 - \delta\bigr)\,.$$

From this it is easy to derive an $O(n)$ procedure that updates $\nabla W(\alpha)$ and calculates $S(\alpha)$. Procedure 2 provides pseudocode for this task.

Procedure 2 Update $\nabla W(\alpha)$ in direction i by δ and calculate $S(\alpha)$

```
$T(\alpha) \leftarrow T(\alpha) - \delta\bigl(2\nabla W_i(\alpha) - 1 - \delta\bigr)$
$E(\alpha) \leftarrow 0$
for $j = 1$ to $n$ do
    $\nabla W_j(\alpha) \leftarrow \nabla W_j(\alpha) - \delta K_{i,j}$
    $E(\alpha) \leftarrow E(\alpha) + C_i \cdot [\nabla W_i(\alpha)]_0^2$
end for
$S(\alpha) \leftarrow T(\alpha) + E(\alpha)$
```

Now the basic idea of the 1D-SVM described in Algorithm 1 is to repeatedly look for the best direction i^* and update in this direction until the stopping criterion (9) is satisfied. However, a closer look at this algorithm shows that it contains one piece of pseudo-code that has not been discussed so far, namely the initialization of the solver. This initialization will be considered in the following subsection.

Algorithm 1 1D-SVM solver

```
initialize $\alpha$, $\nabla W(\alpha)$, $T(\alpha)$, and $S(\alpha)$ by one of the Procedures from Section 2.3
while $S(\alpha) > \frac{\varepsilon}{2\lambda}$ do
    $i^* \leftarrow \arg\max_{i=1,\dots,n} W(\alpha + \delta_i e_i) - W(\alpha)$
    $\delta \leftarrow [\nabla W_{i^*}(\alpha) + \alpha_{i^*}]_0^C - \alpha_{i^*}$
    $\alpha_{i^*} \leftarrow [\nabla W_{i^*}(\alpha) + \alpha_{i^*}]_0^C$
    use Procedure 2 to update $\nabla W(\alpha)$ in direction $i^*$ by $\delta$ and calculate $S(\alpha)$
end while
```

2.3 Initialization

We also have to decide how to initialize α. Of course, there exist various approaches for this task, and in the following, we describe a few methods we have considered in this work.

I0 & W0: Cold Start With Zeros. Obviously, the most simple initialization is the *cold start* $\alpha \leftarrow 0$. Procedure 3 provides the pseudocode for this approach, which in the following we call I0 or W0.

Procedure 3 Initialize by $\alpha_i \leftarrow 0$ and compute $\nabla W(\alpha)$, $S(\alpha)$, and $T(\alpha)$.

```
$T(\alpha) \leftarrow 0$
$S(\alpha) \leftarrow 0$
for $i = 1$ to $n$ do
    $\alpha_i \leftarrow 0$
    $\nabla W_i(\alpha) \leftarrow 1$
    $S(\alpha) \leftarrow S(\alpha) + C_i$
end for
```

I1 & W1: Cold Start With Kernel Rule. Another simple cold start is to initialize with $\alpha_i \leftarrow C_i$ for all $i = 1,\dots,n$. Procedure 4 provides the pseudocode for this approach. In the following, we call this approach I1 or W1.

Procedure 4 Initialize by $\alpha_i \leftarrow C_i$ and compute $\nabla W(\alpha)$, $S(\alpha)$, and $T(\alpha)$.

```
$T(\alpha) \leftarrow 0$
$E(\alpha) \leftarrow 0$
for $i = 1$ to $n$ do
    $\alpha_i \leftarrow C_i$
    $\nabla W_i(\alpha) \leftarrow 1$
    for $j = 1$ to $n$ do
        $\nabla W_i(\alpha) \leftarrow \nabla W_i(\alpha) - C_j \cdot K_{i,j}$
    end for
    $T(\alpha) \leftarrow T(\alpha) - C_i \cdot \nabla W_i(\alpha)$
    $E(\alpha) \leftarrow E(\alpha) + C_i \cdot [\nabla W_i(\alpha)]_0^2$
end for
$S(\alpha) \leftarrow T(\alpha) + E(\alpha)$
```

Obviously, Procedure 3 is $O(n)$, whereas Procedure 4 is $O(n^2)$, and hence the latter seems to be prohibitive. On the other hand, Procedure 4 basically initializes with the classical kernel rule, see

(Devroye et al., 1996, Chapter 10), and hence its initial training error may be significantly smaller than that of Procedure 4. This in turn might lead to a smaller initial stopping criterion value $S(\alpha)$ and hence to less iterations of the solver. Of course, here is a lot of room for speculation, and hence we need to investigate the efficiency of both approaches in the experiments. However, it is worth noting that unlike Procedure 3, Procedure 4 cannot be directly implemented for SVMs *with* offsets. In addition, Procedure 4 requires the entire kernel matrix to be computed, and hence it may actually be prohibitive if this matrix does not fit into memory.

W2: Warm Start By Recycling Old Solution. Besides the cold starts mentioned above, there are also a couple of simple *warm starts* possible. To explain these, let us recall that often the hyperparameter λ is chosen by a search over a grid $\Lambda = \{\lambda_1, \dots, \lambda_m\}$ of candidate values. Let us assume that these values are ordered in the form $\lambda_1 > \dots > \lambda_m$, and that we train the SVM in the order $\lambda_1, \dots, \lambda_m$. Then the resulting n-dimensional vectors $C^{(1)}, \dots, C^{(m)}$ defined by

$$C_i^{(j)} := \begin{cases} \frac{w_{\text{pos}}}{2\lambda_j n} & \text{if } y_i = 1 \\ \frac{w_{\text{neg}}}{2\lambda_j n} & \text{if } y_i = -1 \end{cases}$$

have the property $C_i^{(j)} < C_i^{(j+1)}$ for all $j = 1, \dots, m-1$ and $i = 1, \dots, n$. For $C^{(1)}$ we can then initialize with one of the above cold starts. Now observe that for $j \geq 2$ the approximate solution α^* obtained by training with $C^{\text{old}} := C^{(j-1)}$ is feasible for $C^{\text{new}} := C^{(j)}$, that is, $\alpha^* \in [0, C^{\text{new}}]$. Consequently, for $j \geq 2$ we can either initialize with a cold start, or with the warm start $\alpha \leftarrow \alpha^*$. Obviously, in this case we can also recycle $\nabla W(\alpha)$ and $T(\alpha)$. In addition, the ratio

$$\frac{C_i^{\text{new}}}{C_i^{\text{old}}} = \frac{\lambda_{j-1}}{\lambda_j}$$

is *independent* of i and hence this warm start can be very easily implemented as Procedure 5 shows.

Procedure 5 Initialize by $\alpha_i \leftarrow \alpha_i^*$ and compute $\nabla W(\alpha)$, $S(\alpha)$, and $T(\alpha)$.

```
    $S(\alpha) \leftarrow T(\alpha^*) + \frac{C_1^{\text{new}}}{C_1^{\text{old}}} \cdot (S(\alpha^*) - T(\alpha^*))$
```

W4: Warm Start By Partially Expanding And Partially Recycling Old Solution. Apart from the simple warm start above there is another conceptionally simple warm start for expanding box constraints. Namely, if α^* denotes an approximate solution to C^{old} and $C^{\text{old}} < C^{\text{new}}$ this warm start initializes by $\alpha_i \leftarrow \alpha_i^*$ if $\alpha_i^* < C_i^{\text{old}}$ and by $\alpha_i \leftarrow C_i^{\text{new}}$ if $\alpha_i^* = C_i^{\text{old}}$. The idea behind this warm start is that *bounded* support vectors, that is, indices in

$$bSV := \{j : \alpha_j^* = C_j^{\text{old}}\}$$

may have the tendency to become larger, when the box constraint is loosened, while *unbounded* support vectors, that is, vectors in

$$uSV := \{j : 0 < \alpha_j^* < C_j^{\text{old}}\}$$

may not have this tendency.

The basic idea of an efficient implementation of this warm start method is to avoid calculating the gradient from scratch by recycling parts of the gradient from C^{old}. To be more precise, observe that, for fixed i, the sum $\sum_{j \in uSV} \alpha_j^* K_{i,j}$ remains unchanged by the described warm start, while

Procedure 6 Initialize bounded SVs by $\alpha_i \leftarrow C_i^{\text{new}}$ while keeping the rest unchanged and compute $\nabla W(\alpha)$, $S(\alpha)$, and $T(\alpha)$.

```
$T(\alpha) \leftarrow 0$
$E(\alpha) \leftarrow 0$
for $i = 1$ to $n$ do
    if $\alpha_i = C_i^{\text{old}}$ then
        $\alpha_i \leftarrow C_i^{\text{new}}$
    end if
end for
if $2 \cdot \#uSV < \#bSV$ then
    for $i = 1$ to $n$ do
        $\nabla W_i(\alpha) \leftarrow \frac{C_1^{\text{new}}}{C_1^{\text{old}}} \cdot \nabla W_i(\alpha) + \big(1 - \frac{C_1^{\text{new}}}{C_1^{\text{old}}}\big)\big(1 - \sum_{j \in uSV} \alpha_j K_{i,j}\big)$
        $T(\alpha) \leftarrow T(\alpha) - \alpha_i \cdot \nabla W_i(\alpha)$
        $E(\alpha) \leftarrow E(\alpha) + C_i^{\text{new}} \cdot [\nabla W_i(\alpha)]_0^2$
    end for
else
    for $i = 1$ to $n$ do
        $\nabla W_i(\alpha) \leftarrow \nabla W_i(\alpha) + (C_i^{\text{old}} - C_i^{\text{new}}) \sum_{j \in bSV} K_{i,j}$
        $T(\alpha) \leftarrow T(\alpha) - \alpha_i \cdot \nabla W_i(\alpha)$
        $E(\alpha) \leftarrow E(\alpha) + C_i^{\text{new}} \cdot [\nabla W_i(\alpha)]_0^2$
    end for
end if
$S(\alpha) \leftarrow T(\alpha) + E(\alpha)$
```

$\sum_{j \in bSV} \alpha_j^* K_{i,j}$ is simply multiplied by $C_i^{\text{new}}/C_i^{\text{old}}$. Recall that the latter ratio is independent of i, and consequently we can update the gradients by either

$$\nabla W_i(\alpha) \leftarrow 1 - \frac{C_1^{\text{new}}}{C_1^{\text{old}}} \left(1 - \nabla W_i(\alpha^*) - \sum_{j \in uSV} \alpha_j^* K_{i,j}\right) - \sum_{j \in uSV} \alpha_j^* K_{i,j}$$

for all $i = 1, \ldots, n$, or

$$\nabla W_i(\alpha) \leftarrow \nabla W_i(\alpha) + (C_i^{\text{old}} - C_i^{\text{new}}) \sum_{j \in bSV} K_{i,j}\,, \qquad i = 1, \ldots, n\,,$$

where in the first formula we used

$$1 - \nabla W_i(\alpha^*) - \sum_{j \in uSV} \alpha_j^* K_{i,j} = \sum_{j \in bSV} \alpha_j^* K_{i,j}\,. \tag{10}$$

Note that the first method implicitly recycles $\sum_{j \in bSV} \alpha_j^* K_{i,j}$ by (10), while the second method implicitly recycles $\sum_{j \in uSV} \alpha_j^* K_{i,j}$. Obviously, depending on the number of bounded and unbounded support vectors either the first or the second method is more efficient, and hence should be chosen. We decided to pick the first or second method depending on whether $2 \cdot \#uSV < \#bSV$ or not. This decision was based on counts of the involved floating point operations and the fact that in all our experiments we stored the entire kernel matrix in the memory. However note that both methods require to access some rows of the kernel matrix, and hence there is most likely a more efficient cut-off when only parts of the kernel matrix are stored in memory by caching. Since in general, the costs of computing a row of the kernel matrix depends on data set specific features, such as its dimensionality when using Gaussian kernels, there does not seem to exists a simple rule of thumb in this case, though. Consequently, we decided not to analyze this case carefully. Procedure 6 displays the corresponding pseudocode for this warm start, which we call W4. It is not hard to see, that in the worst case Procedure 6 is $O(n^2)$, while in the best case it is only $O(n)$. Since the average case cannot be easily analyzed, we need to experimentally evaluate whether this warm start is efficient or not.

W6: Warm Start By Partially Shrinking And Partially Recycling Old Solution. Let us now assume that we run through the λ-grid in reverse order. Then we have $C^{\text{old}} > C^{\text{new}}$, and hence we

Procedure 7 Initialize directions that violate the new box constrained by $\alpha_i \leftarrow C_i^{\text{new}}$ while keeping the rest unchanged and compute $\nabla W(\alpha)$, $S(\alpha)$, and $T(\alpha)$.

```
for $i = 1$ to $n$ do
  if $\alpha_i > C_i^{\text{new}}$ then
    $\alpha_i \leftarrow C_i^{\text{new}}$
  end if
end for
$T(\alpha) \leftarrow 0$
$E(\alpha) \leftarrow 0$
if $\#nuSV < \#bSV$ then
  for $i = 1$ to $n$ do
    $\nabla W_i(\alpha) \leftarrow 1 - \frac{C_1^{\text{new}}}{C_1^{\text{old}}} \cdot \big(1 - \nabla W_i(\alpha) - \sum_{j \in nuSV} \alpha_j^* K_{i,j} - \sum_{j \in nbSV} \alpha_j^* K_{i,j}\big)$
    $\nabla W_i(\alpha) \leftarrow \nabla W_i(\alpha) - \sum_{j \in nuSV} \alpha_j^* K_{i,j} - \sum_{j \in nbSV} C_j^{\text{new}} K_{i,j}$
    $T(\alpha) \leftarrow T(\alpha) - \alpha_i \cdot \nabla W_i(\alpha)$
    $E(\alpha) \leftarrow E(\alpha) + C_i^{\text{new}} \cdot [\nabla W_i(\alpha)]_0^2$
  end for
else
  for $i = 1$ to $n$ do
    $\nabla W_i(\alpha) \leftarrow \nabla W_i(\alpha) + \sum_{j \in bSV} (C_j^{\text{old}} - C_j^{\text{new}}) K_{i,j}$
    $\nabla W_i(\alpha) \leftarrow \nabla W_i(\alpha) + \sum_{j \in nbSV} (\alpha_j^* - C_j^{\text{new}}) K_{i,j}$
    $T(\alpha) \leftarrow T(\alpha) - \alpha_i \cdot \nabla W_i(\alpha)$
    $E(\alpha) \leftarrow E(\alpha) + C_i^{\text{new}} \cdot [\nabla W_i(\alpha)]_0^2$
  end for
end if
$S(\alpha) \leftarrow T(\alpha) + E(\alpha)$
```

cannot immediately recycle the old approximate solution α^*. Nonetheless, there is a certain analogue to Procedure 6 possible. Indeed, we can initialize by $\alpha_i \leftarrow \alpha_i^*$ if $\alpha_i^* \leq C^{\text{new}}$ and by $\alpha_i \leftarrow C^{\text{new}}$ if $\alpha_i^* > C^{\text{new}}$. Again, the corresponding warm start needs some work to find an efficient implementation that recycles suitable parts of the gradient. In order to explain such an implementation we split the set *uSV* into

$$\begin{aligned} nuSV &:= \{j : 0 < \alpha_j^* \le C_j^{\text{new}}\}, \\ nbSV &:= \{j : C_j^{\text{new}} < \alpha_j^* < C_j^{\text{old}}\}, \end{aligned}$$

where we note that we use a slight abuse of the letters *u* and *b* in this notation. Now note that the initialization above multiplies all $\alpha_j^* \in bSV$ by the factor $C_1^{\text{new}}/C_1^{\text{old}}$, while it keeps all $\alpha_j^* \in nuSV$ unchanged. Obviously, both update rules make it possible to recycle parts of the gradient. Unfortunately, however, for $\alpha_j^* \in nbSV$, the situation is more complicated and no simple recycling is possible. Thus, Procedure 7, which displays the corresponding pseudocode, is a little more complicated than Procedure 6. Nonetheless, all remarks concerning the computational requirements of Procedure 6 also apply to Procedure 7, and the same holds true for the rule that decides which part of the gradient is recycled. In the following, we call this approach displayed in Procedure 7, W6.

W3 & W5: Warm Start By Scaling Old Solution. Finally, there is an easy warm start option that works regardless of the direction we run through the λ-grid. Indeed, we can always initialize by $\alpha_i \leftarrow \alpha_i^* \cdot C_1^{\text{new}}/C_1^{\text{old}}$. The Procedure 8 shows the corresponding $O(n)$ pseudocode. Depending on whether $C_1^{\text{old}} < C_1^{\text{new}}$ or $C_1^{\text{old}} > C_1^{\text{new}}$ we call this approach W3 or W5, respectively.

Procedure 8 Initialize by $\alpha_i \leftarrow \alpha_i^* \cdot C_1^{\text{new}}/C_1^{\text{old}}$ and compute $\nabla W(\alpha)$, $S(\alpha)$, and $T(\alpha)$.

```
$T(\alpha) \leftarrow 0$
$E(\alpha) \leftarrow 0$
**for** $i = 1$ to $n$ **do**
    $\alpha_i \leftarrow \frac{C_1^{\text{new}}}{C_1^{\text{old}}} \cdot \alpha_i^*$
    $\nabla W_i(\alpha) \leftarrow 1 - \frac{C_1^{\text{new}}}{C_1^{\text{old}}} \cdot \big(1 - \nabla W_i(\alpha)\big)$
    $T(\alpha) \leftarrow T(\alpha) - \alpha_i \cdot \nabla W_i(\alpha)$
    $E(\alpha) \leftarrow E(\alpha) + C_i^{\text{new}} \cdot [\nabla W_i(\alpha)]_0^2$
**end for**
$S(\alpha) \leftarrow T(\alpha) + E(\alpha)$
```

3. Working Sets of Size Two

So far, our algorithm performs an update in one coordinate per iteration. Let us now consider an algorithm which performs an update in *two* coordinates per iteration. To this end, let us first present the following, simple lemma that computes the gain of a 2-dimensional update.

Lemma 2 *For $\delta_i, \delta_j \in \mathbb{R}$ and $i, j = 1, \ldots, n$ we have*

$$W(\alpha + \delta_i e_i + \delta_j e_j) - W(\alpha) = \delta_i \cdot (\nabla W_i(\alpha) - \delta_i/2) + \delta_j \cdot (\nabla W_j(\alpha) - \delta_j/2) - \delta_i \delta_j K_{i,j}\,.$$

Proof Applying Lemma 1 twice and using the formula $\nabla W_j(\alpha + \delta_i e_i) = \nabla W_j(\alpha) - \delta_i K_{i,j}$ we find the assertion. ■

3.1 Solving the Two-Dimensional Problem Exactly

In order to describe an algorithm that updates two variables at each iteration we first have to investigate how the two-variable update looks like in detail. To this end, we fix two coordinates $i,j\in\{1,\dots,n\}$ with $i\neq j$ and consider the function

$$(\tilde{\alpha}_i,\tilde{\alpha}_j)\mapsto W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_j):=W(\alpha^{\setminus i,j}+\tilde{\alpha}_i e_i+\tilde{\alpha}_j e_j)\,,$$

where $\alpha^{\setminus i,j}:=\alpha-\alpha_i e_i-\alpha_j e_j$ is a fixed vector whose i-th and j-th coordinates equal zero. A simple calculation then shows

$$\begin{aligned} W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_j) &= \langle e,\alpha^{\setminus i,j}\rangle+\tilde{\alpha}_i+\tilde{\alpha}_j-\frac{1}{2}\langle\alpha^{\setminus i,j},K\alpha^{\setminus i,j}\rangle-\tilde{\alpha}_i\langle e_i,K\alpha^{\setminus i,j}\rangle-\tilde{\alpha}_j\langle e_j,K\alpha^{\setminus i,j}\rangle \\ &\quad -\frac{1}{2}\big(\tilde{\alpha}_i^2+2\tilde{\alpha}_i\tilde{\alpha}_jK_{i,j}+\tilde{\alpha}_j^2\big)\,, \end{aligned}$$

where we used $K_{i,i}=K_{j,j}=1$. Consequently, the partial derivatives are given by

$$\begin{aligned} \frac{\partial W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_j)}{\partial\tilde{\alpha}_i} &= 1-\langle e_i,K\alpha^{\setminus i,j}\rangle-\tilde{\alpha}_i-\tilde{\alpha}_jK_{i,j}\,, \\ \frac{\partial W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_j)}{\partial\tilde{\alpha}_j} &= 1-\langle e_j,K\alpha^{\setminus i,j}\rangle-\tilde{\alpha}_j-\tilde{\alpha}_iK_{i,j}\,. \end{aligned}$$

In order to derive the maximum of $W_{i,j}$ on $[0,C_i]\times[0,C_j]$ from these derivatives, we need to consider three different cases.

The Case $K_{i,j}=1$. By setting the above derivatives to zero, we obtain the following system of linear equations

$$\begin{aligned} \alpha_i^*+\alpha_j^* &= 1-\langle e_i,K\alpha^{\setminus i,j}\rangle\,, \\ \alpha_i^*+\alpha_j^* &= 1-\langle e_j,K\alpha^{\setminus i,j}\rangle \end{aligned}$$

that have to be satisfied for all global maxima $(\alpha_i^*,\alpha_j^*)\in\mathbb{R}^2$ of $W_{i,j}$. Now recall that we assumed that the kernel k is strictly positive definite, and therefore we see that $K_{i,j}=1$ implies $x_i=x_j$, and hence $y_i=y_j$. From this we conclude $K_{i,\ell}=K_{j,\ell}$ for all $\ell=1,\dots,n$, and thus we obtain $1-\langle e_i,K\alpha^{\setminus i,j}\rangle=1-\langle e_j,K\alpha^{\setminus i,j}\rangle$. Consequently, $W_{i,j}$ attains its global maximum at every point of the affine subspace

$$\big\{(\alpha_i^*,\alpha_j^*):\alpha_i^*+\alpha_j^*=1-\langle e_i,K\alpha^{\setminus i,j}\rangle\big\}\,, \tag{11}$$

which is a translated version of the anti-diagonal subspace $\{(\alpha,-\alpha):\alpha\in\mathbb{R}\}$.

Now recall that $y_i=y_j$ implies $C_i=C_j$, and hence we are actually interested in finding a pair $(\tilde{\alpha}_i,\tilde{\alpha}_j)$ that maximizes $W_{i,j}$ on the square $[0,C_i]^2$. If $1-\langle e_i,K\alpha^{\setminus i,j}\rangle\in[0,2C_i]$, it is easy to see that the subspace (11) intersects the square, and hence $W_{i,j}$ attains the desired maximum at every element in this intersection. In particular, (α_i^*,α_i^*), where

$$\alpha_i^*:=\frac{1-\langle e_i,K\alpha^{\setminus i,j}\rangle}{2}$$

is such a pair. Let us now assume that $1-\langle e_i,K\alpha^{\setminus i,j}\rangle>2C_i$. Then the subspace (11) lies “above” the square $[0,C_i]^2$, and since $W_{i,j}$ is concave, $W_{i,j}$ then attains its maximum over $[0,C_i]^2$ at a point of the set of edges $\{C_i\}\times[0,C_i]\cup[0,C_i]\times\{C_i\}$. Let us fix a pair $(\tilde{\alpha}_i,\tilde{\alpha}_j)\in\{C_i\}\times[0,C_i]$. Then we have

$$\frac{\partial W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_j)}{\partial\tilde{\alpha}_j}=1-\langle e_j,K\alpha^{\setminus i,j}\rangle-\tilde{\alpha}_j-\tilde{\alpha}_iK_{i,j}=1-\langle e_j,K\alpha^{\setminus i,j}\rangle-\tilde{\alpha}_j-C_i>0\,,$$

and hence $W_{i,j}$ attains its maximum over $\{C_i\}\times[0,C_i]$ at the corner (C_i,C_i). Interchanging the roles of i and j we can thus conclude that $W_{i,j}$ attains its maximum over $[0,C_i]^2$ at (C_i,C_i). Since we can analogously show that, for $1-\langle e_i,K\alpha^{\setminus i,j}\rangle<0$, the function $W_{i,j}$ attains its maximum over $[0,C_i]^2$ at $(0,0)$, we finally find the update rule

$$\alpha_i^{new}:=\alpha_j^{new}:=\left[\frac{1-\langle e_i,K\alpha^{\setminus i,j}\rangle}{2}\right]_0^{C_i}=\left[\frac{\nabla W_i(\alpha)+\alpha_i+\alpha_j}{2}\right]_0^{C_i}.$$

The Case $K_{i,j}=-1$. In this case, we have $x_i=x_j$, and hence $y_i=-y_j$. From this we conclude $K_{i,\ell}=-K_{j,\ell}$ for all $\ell=1,\dots,n$, and thus we obtain $\langle e_i,K\alpha^{\setminus i,j}\rangle=-\langle e_j,K\alpha^{\setminus i,j}\rangle$. Consequently, the derivatives above reduce to

$$\begin{aligned}\frac{\partial W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_j)}{\partial\tilde{\alpha}_i}&=1-\langle e_i,K\alpha^{\setminus i,j}\rangle-\tilde{\alpha}_i+\tilde{\alpha}_j\,,\\ \frac{\partial W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_j)}{\partial\tilde{\alpha}_j}&=1+\langle e_i,K\alpha^{\setminus i,j}\rangle-\tilde{\alpha}_j+\tilde{\alpha}_i\,,\end{aligned}$$

and from this it is easy to conclude that $W_{i,j}$ does not have a global maximum. However, a closer inspection of $W_{i,j}$ yields the formula

$$W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_j)=\langle e,\alpha^{\setminus i,j}\rangle+\tilde{\alpha}_i+\tilde{\alpha}_j-\frac{1}{2}\langle\alpha^{\setminus i,j},K\alpha^{\setminus i,j}\rangle-(\tilde{\alpha}_i-\tilde{\alpha}_j)\langle e_i,K\alpha^{\setminus i,j}\rangle-\frac{1}{2}\big(\tilde{\alpha}_i-\tilde{\alpha}_j\big)^2\,,$$

and hence we see that, for fixed $\beta\in\mathbb{R}$, we have

$$W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_i+\beta)=\langle e,\alpha^{\setminus i,j}\rangle+2\tilde{\alpha}_i+\beta-\frac{1}{2}\langle\alpha^{\setminus i,j},K\alpha^{\setminus i,j}\rangle+\beta\langle e_i,K\alpha^{\setminus i,j}\rangle-\frac{1}{2}\beta^2\,.$$

In other words, $W_{i,j}$ is a affine linear function with positive slope on the affine subspaces

$$\{(\tilde{\alpha}_i,\tilde{\alpha}_i+\beta):\tilde{\alpha}_i\in\mathbb{R}\}\,,\qquad\qquad\beta\in\mathbb{R},$$

and therefore $W_{i,j}$ attains its maximum over $[0,C_i]\times[0,C_j]$ at a point from the set of edges $\{C_i\}\times[0,C_j]\cup[0,C_i]\times\{C_j\}$. Let us first consider a pair $(\tilde{\alpha}_i,\tilde{\alpha}_j)\in\{C_i\}\times[0,C_j]$. Then we have

$$\frac{\partial W_{i,j}(\tilde{\alpha}_i,\tilde{\alpha}_j)}{\partial\tilde{\alpha}_j}=1-\langle e_j,K\alpha^{\setminus i,j}\rangle-\tilde{\alpha}_j+C_i\,,$$

and hence $W_{i,j}$ attains its maximum over $\{C_i\}\times[0,C_j]$ at (C_i,α_j^*), where

$$\alpha_j^*=[1-\langle e_j,K\alpha^{\setminus i,j}\rangle+C_i]_0^{C_j}=[\nabla W_j(\alpha)+\alpha_j-\alpha_i+C_i]_0^{C_j}\,.$$

Moreover, for $\delta_i:=C_i-\alpha_i$ and $\delta_j:=\alpha_j^*-\alpha_j$ we obtain the gain of this update by Lemma 2. Analogously, we can show that $W_{i,j}$ attains its maximum over $[0,C_i]\times\{C_j\}$ at (α_i^*,C_j), where

$$\alpha_i^*=[1-\langle e_i,K\alpha^{\setminus i,j}\rangle+C_j]_0^{C_i}=[\nabla W_i(\alpha)+\alpha_i-\alpha_j+C_j]_0^{C_i}\,.$$

Again, the gain of the corresponding update can be computed by Lemma 2, and by comparing both gains we can then decide which two-dimensional update yields the larger gain. The corresponding update is chosen in the algorithm.

The Case $K_{i,j} \neq \pm 1$. To solve the two dimensional problem in this case we fix an $\alpha \in \mathbb{R}^n$ and write

$$\begin{aligned}
\gamma_i &:= 1 - \langle e_i, K\alpha^{\setminus i,j} \rangle = 1 - \sum_{\ell \neq i,j} \alpha_\ell K_{i,\ell} = \nabla W_i(\alpha) + \alpha_i + \alpha_j K_{i,j}\,, \\
\gamma_j &:= 1 - \langle e_j, K\alpha^{\setminus i,j} \rangle = 1 - \sum_{\ell \neq i,j} \alpha_\ell K_{j,\ell} = \nabla W_j(\alpha) + \alpha_j + \alpha_i K_{i,j}\,.
\end{aligned}$$

Using the derivatives of $W_{i,j}$ it is then easy to see that $W_{i,j}$ attains its global maximum at each point (α_i^*, α_j^*) that satisfies $\gamma_i = \alpha_i^* + \alpha_j^* K_{i,j}$ and $\gamma_j = \alpha_j^* + \alpha_i^* K_{i,j}$. Furthermore, simple algebraic transformations show

$$\alpha_i^* = \frac{\gamma_i - \gamma_j K_{i,j}}{1 - K_{i,j}^2} \qquad \text{and} \qquad \alpha_j^* = \frac{\gamma_j - \gamma_i K_{i,j}}{1 - K_{i,j}^2}\,,$$

and by re-substituting the definition of γ_i and γ_j we hence obtain

$$\begin{aligned}
\alpha_i^* &= \alpha_i + \tfrac{\nabla W_i(\alpha) - \nabla W_j(\alpha) K_{i,j}}{1 - K_{i,j}^2}\,, \\
\alpha_j^* &= \alpha_j + \tfrac{\nabla W_j(\alpha) - \nabla W_i(\alpha) K_{i,j}}{1 - K_{i,j}^2}
\end{aligned} \tag{12}$$

for the uniquely determined point at which $W_{i,j}$ attains its global maximum. Now if $(\alpha_i^*, \alpha_j^*) \in [0, C_i] \times [0, C_j]$ we can simply update by $(\alpha_i^{new}, \alpha_j^{new}) := (\alpha_i^*, \alpha_j^*)$. However, if $(\alpha_i^*, \alpha_j^*) \notin [0, C_i] \times [0, C_j]$ we have to make further calculations. For example, for $\alpha_i^* > C_i$ and $\alpha_j^* \in [0, C_j]$, the function $W_{i,j}$ attains its maximum over $[0, C_i] \times [0, C_j]$ at a point of the line $\{C_i\} \times [0, C_j]$ by the concavity of $W_{i,j}$. Consequently, in this case the update is

$$(\alpha_i^{new}, \alpha_j^{new}) := \big(C_i, [\nabla W_j(\alpha) + (\alpha_i - C_i) K_{i,j} + \alpha_j]_0^{C_j}\big)\,,$$

that is, we first update the i-th coordinate, which leads to the temporary gradient

$$\nabla W_j(\alpha) + (\alpha_i - C_i) K_{i,j}\,,$$

and then perform a one-dimensional optimization over the j-th coordinate. The remaining three cases where exactly one direction of (α_i^*, α_j^*) violates the box constraint can be handled analogously. Finally, let us consider the cases, where both coordinates violate the constraint, for example, $\alpha_i^* > C_i$ and $\alpha_j^* > C_j$. In this case, the concavity of $W_{i,j}$ shows that $W_{i,j}$ attains its maximum over $[0, C_i] \times [0, C_j]$ at a point of the set $\{C_i\} \times [0, C_j] \cup [0, C_i] \times \{C_j\}$. Consequently, we have to temporarily perform the one-dimensional optimization above twice, namely one over the i-th coordinate and one over the j-th coordinate. By computing the resulting gain of W for both optimizations, we can then decide which optimization we have to choose for the update. Again, the remaining three cases can be handled analogously.

Algorithm 2 2D-SVM solver

```
initialize $\big(\alpha, \nabla W(\alpha), T(\alpha), S(\alpha)\big)$
while $S(\alpha) > \frac{\varepsilon}{2\lambda}$ do
  select directions $i^*$ and $j^*$
  update $\alpha$ in the directions $i^*$ and $j^*$
  update $\nabla W(\alpha)$ in the directions $i^*$ and $j^*$ and calculate $T(\alpha)$ and $E(\alpha)$
  $S(\alpha) \leftarrow T(\alpha) + E(\alpha)$
end while
```

3.2 Selecting a Working Set of Size Two

The 2D-SVM-solver displayed in Algorithm 2 is conceptionally very similar to the 1D-SVM-solver presented in Algorithm 1. However, so far we have not addressed how to choose the directions i^* and j^* in which the 2D-SVM-solver performs an update. Obviously, several possibilities exists for this task, and we discuss a few of them in the following.

WSS 0: Choose The Pair Of Directions With Maximal Gain. Given a pair of directions (i, j), Lemma 2 can be used to compute the gain of W resulting from the exact two dimensional optimization described in Section 3.1. Now one could consider all pairs of directions and choose the one with the largest gain. Of course, in practice this approach is prohibitive, since the search is an $O(n^2)$ operation, which has to be performed in each iteration. Nonetheless, in some sense this approach may be viewed as an “optimal” two dimensional strategy, and all subset selection strategies developed below can be interpreted as low cost approximations to this approach. Consequently, we tested it to get a baseline number of iterations, to which all other subset selection strategies are compared to.

WSS 1: 1D-direction With Maximal Gain And Previously Found 1D-direction. A careful analysis of the behavior of the 1D-SVM-solver shows that it often comes into a regime in which it picks alternating indices i^* and j^* for a while. In other words, it tries to approximately solve the 2D-problem in the directions i^* and j^*. In order to avoid this cost-intensive alternating we can look for the best 1D-direction i^* and then perform a 2D-update over i^* and the 1D-direction i^*_{old} chosen in the previous iteration. Conceptionally, this approach is very close to the maximum-gain procedure mentioned in Glasmachers and Igel (2006) for SVMs with offset. The advantage of this approach is that it preserves the low-cost search from the 1D-SVM-solver. On the downside, however, it may not reduce the number of iterations very effectively.

WSS 2: Two 1D-directions With Maximal Gain From Separate Subsets. Another simple way to preserve the low cost search from the 1D-SVM-solver is to split the index set $\{1, \ldots, n\}$ into two parts $\{1, \ldots, n/2\}$ and $\{n/2+1, \ldots, n\}$ and search for the 1D-directions with maximal gain over these two parts separately. In other words, we can choose the directions i^* and j^* by

$$\begin{aligned} i^* &\in \arg\max_{i \le n/2} W(\alpha + \delta_i e_i) - W(\alpha)\,, \\ j^* &\in \arg\max_{i > n/2} W(\alpha + \delta_i e_i) - W(\alpha)\,, \end{aligned}$$

where δ_i is defined as in the 1D-SVM-solver. Clearly, this approach preserves the low cost search from the 1D-SVM-solver, but again it is not clear whether it reduces the number of iterations very effectively.

WSS 4: 1D-direction With Maximal Gain And A Direction Of A Nearby Sample. Yet another approach to preserve the low cost search from the 1D-SVM-solver is to first look for the 1D-direction i^* with maximal gain, and then, in a second step, to pick a direction j^* such that x_{j^*} is close to x_{i^*} with respect to the metric

$$d_k(x,x') := \sqrt{2-2k(x,x')}, \qquad x,x' \in X,$$

induced by the kernel. Note that x is close to x' in this metric, if and only if $k(x,x')$ is close to 1. Consequently, the gradients of the samples close to x_{i^*} are the ones that are most affected by an update in direction i^*. Therefore, if these gradients are close to zero *before* the update, they will most likely be no longer close to zero *after* the update, and hence the corresponding directions will have a good chance of being chosen in a subsequent iteration. In our experiments, we considered the k-nearest neighbors of x_i^*, where $k = 10$, and picked the neighbor x_{j^*} for which the 2D-update in the directions (i^*, j^*) yielded the largest gain. Note that, as soon as the direction i^* is found, it is clear that one subsequently needs to access the i^*-th kernel row for updating the gradient. Therefore, this working set selection strategy does not require further kernel computations. Moreover, computing the 2D-gain over k candidates is also relatively inexpensive, if k remains small. Nonetheless, initial experiments suggested that searching over the k-nearest neighbors only makes sense when the solver mainly updates *inner* support vectors, that is, directions i with $0 < \alpha_i < C_i$. Consequently, we implemented a Boolean flag that was recomputed every 10 iterations. In this re-computation, the flag was set to true, if and only if in at least 5 of the previous 10 iterations the picked directions i^* and j^* both were inner support vectors. We then considered the k-nearest neighbors only if this Boolean flag was set, while in the other case we applied the working set selection strategy WSS 1.

WSS x: Combinations Of 1D-direction-based Approaches. It is easy to see that one can combine the previous three methods that are based on finding the 1D-direction with maximal gain. For example, in each iteration one can combine WSS 1 and WSS 2 by computing the 2D-gain of both methods and pick the one with the larger gain. Obviously, this still preserves the low cost search from the 1D-SVM-solver and only adds little cost for computing the 2D-gain for the two candidate pairs. Similarly, all three methods can be combined. Combinations of these methods are called WSS x, where x is the sum of the combined methods. For example, by combining WSS 1, WSS 2, and WSS 4 we obtain WSS 7, and by combining WSS 1, WSS 2, WSS 4 with WSS 512 below, we obtain WSS 519 . In the following, we keep this binary numbering system which makes it possible to easily describe arbitrary combinations of basic working set selection strategies.

WSS 8: 1D-direction With Maximal Gain And One-step-ahead 1D-direction. Another way to extend the 1D-SVM subset selection strategy to two directions is to first look for the 1D-direction i^* with maximal gain, and then to look for the 1D-direction j^* with maximal gain that would be found after having updated in direction i^*. Obviously, this strategy, which we call WSS 8, is closely related to WSS 1 in that the update and search routines are partially permuted. However, it has a higher cost for the search part per iteration, while intuitively it should reduce the number of iterations.

WSS 16: Maximal Violating Pair. A completely different subset selection strategy is based on the maximal violating pair (MVP) idea, see Keerthi et al. (2001) and Joachims (1999). For the SVM without offset, this means that the pair (i^*, j^*) is chosen that violates (5) most. In other words, for both index sets $\{i : \alpha_i < C_i\}$ and $\{i : \alpha_i > 0\}$ the two indices with the largest, respectively smallest, gradients are picked, and the final pair (i^*, j^*) consists of the indices that have the gradient with the largest absolute value among the four candidate directions. In order to implement this working set selection strategy efficiently, the sets $\{i : \alpha_i < C_i\}$ and $\{i : \alpha_i > 0\}$ should be kept in memory and updated in every iteration. This may add some cost per iteration compared to the previous working set selection strategies, while it is unclear how the number of iterations behave compared to these strategies.

WSS 32: 1D-direction With Maximal Gain And Corresponding "Optimal" 2D-direction. None of the methods introduced so far try to seriously approximate the 2D-subset selection strategy WSS 0, which intuitively picks the best possible pair of indices. The first method that seriously strives for such an approximation is WSS 32, which first picks the 1D-direction i^* with maximal gain, and then searches for the $j^* \in \{1,\dots,n\}$ such that (i^*, j^*) maximizes the corresponding 2D-gain. Obviously, the cost for this search method is significantly higher than those of WSS 1 to WSS 7, but it is still $O(n)$. On the other hand, the better choice of (i^*, j^*) may substantially reduce the number of iterations of the 2D-SVM-solver, and hence it is not a-priori clear how WSS 32 performs compared to the earlier methods. Finally, note that WSS 32 is related to the second order working set selection strategy of Fan et al. (2005), which was proposed for SVMs with offset.

WSS 64: 1D-direction With Maximal Gain And Random "Optimal" 2D-direction. Instead of considering all pairs (i^*, j), $j = 1,\dots,n$, as WSS 32 does, it may suffice to reduce the search over the pairs (i^*, j), $j \in J$, where $J \subset \{1,\dots,n\}$ is a random subset. In our experiments we considered the case $\#J = n/5$.

WSS 128: 1D-direction With Maximal Gain And Approximately "Optimal" 2D-direction. One of the disadvantages of WSS 32 is that computing the 2D-gain is quite expensive due to the relatively large number of branches and floating point operations. One way to address this issue is to compute the 2D-gain in WSS 32 only approximately. WSS 128 uses the following approximation: for indices i and j with $K_{i,j} = \pm 1$ it computes the exact gain, while for the other pairs it first computes α_i^* and α_j^* by (12), and then applies the simple clipping operation

$$\begin{aligned} \alpha_i^{new} &:= [\alpha_i^*]_0^{C_i}, \\ \alpha_j^{new} &:= [\alpha_j^*]_0^{C_j}. \end{aligned}$$

For these new α's, WSS 128 finally computes the gain by Lemma 2. Clearly, this gain is in general less than the exact gain, but it still may be a good approximation. In particular, if both α_i^* and α_j^* satisfy the box constraints, then the approximation is actually exact. On the other hand, the approximation is clearly less expensive, but we expect more iterations compared to WSS 32.

WSS 256: Random 2D-directions With Maximal Gain. Another way to approximate WSS 0 is to consider k random pairs (i, j), and pick the pair (i^*, j) that yields the largest exact 2D-gain among them. In WSS 256 we followed this idea for $k := n$.

WSS 512: 1D-direction With Maximal Gain And 2D-direction Over Inner SVs. Although the approximate computation of the 2D-gain in WSS 128 is cheaper than the exact computation in WSS 32, it may still be too expensive. One way to further decrease these costs is based on the observation that the 2D-gain is given by

$$\frac{1}{2} \cdot \frac{|\nabla W_i(\alpha)|^2 + |\nabla W_j(\alpha)|^2 - 2\nabla W_i(\alpha)\nabla W_j(\alpha)K_{i,j}}{1 - K_{i,j}^2}$$

if $K_{i,j} \neq \pm 1$ and α_i^* and α_j^* computed by (12) satisfy the box constraints. WSS 512 uses this simplified formula in the following way. Again, it first searches for the 1D-direction i^* with maximal gain. If α_{i^*} is an inner support vector, see WSS 4 for a definition, and the Boolean flag of WSS 4 is set, WSS 512 searches for the direction

$$j^* \in \{j : 0 < \alpha_j < C_j \text{ and } K_{i^*,j} \neq \pm 1\}$$

that optimizes the above formula of the 2D-gain for fixed $i := i^*$. Since in some iterations WSS 512 reduces to the 1D-SVM-solver we further considered some combinations with WSS 3, and WSS 7 in our experiments. Following the naming convention of combinations mentioned earlier, these strategies are called WSS 515 and WSS 519.

WSS 1024: 1D-direction With Maximal Gain And Random 2D-direction Over Inner SVs. The next subset selection strategy, WSS 1024, is quite similar to WSS 512, except that it does not consider all inner support vectors in the search for j^*, but only k random inner support vectors. In our experiments we used the k that equaled 20% of the current number of inner support vectors. In addition, we initiated the search whenever α_{i^*} was an inner support vector, that is, the search was initiated *independently* of the Boolean flag of WSS 4. Again, in some iterations WSS 1024 reduces to the 1D-SVM-solver, and hence we further considered some combinations with WSS 1, WSS 2, and WSS 4, where again the naming convention above was used.

WSS 2048: Add Random 2D-directions Over Inner SVs. The final subset selection strategy, WSS 2048, is actually not a subset selection strategy of its own, but only a strategy that works in combination with others. Once one of the previous subset selection strategies has picked a pair (i^*, j^*) and α_{i^*} has turned out to be an inner support vector, WSS 2048 considers k random pairs of inner support vectors, and picks the pair (i^{**}, j^{**}) that has largest approximate gain, where the approximation was computed as in WSS 512. Then the exact gain of (i^*, j^*) and (i^{**}, j^{**}) is computed and the pair with the larger exact gain was chosen. We considered this method in combination with WSS 1, WSS 2, and WSS 4, where again the naming convention above was used.

4. Convergence Analysis

In this section we establish an upper bound on the number of iterations for both the 1D-SVM and the 2D-SVM. Our approach is heavily based on earlier ideas[2] developed for the analysis of rate-certifying decomposition algorithms, see, for example, Hush and Scovel (2003), List and Simon (2005), Hush et al. (2006) and List and Simon (2007), but it may be possible to partially use results on block coordinate descent algorithms such as the one by Luo and Tseng (1992) for the analysis, instead.[3]

Let us begin by recalling from the first papers mentioned that the σ-functional for a vector $\alpha \in [0,C] = [0,C_1] \times \cdots \times [0,C_n]$ and an index set $I \subset \{1,\ldots,n\}$ is defined by

$$\sigma(\alpha|I) := \sup_{\substack{\tilde\alpha\in[0,C] \\ \tilde\alpha_i=\alpha_i \forall i\notin I}} \left\langle \nabla W(\alpha), \tilde\alpha - \alpha \right\rangle .$$

[2] Despite this, we decided to include the analysis, since: *a)* it still requires a little work and thus we felt that it was a bit unfair to the reader to simply say that the analysis is straightforward; *b)* we thought that it was nice to see how the relatively complicated techniques for the offset case significantly simplify; *c)* our goal was to provide a full and self-contained work for the proposed algorithm.

[3] Note, however, that their results only control the convergence to a *dual* optimal solution, while for statistical reasons, we are actually interested in the convergence control of the corresponding primal sequence. Consequently, their results are at least not directly applicable.

Since our algorithms are based on gain optimization rather than rate certification, we further need the γ-functional

$$\gamma(\alpha|I) := \sup_{\substack{\tilde\alpha\in[0,C]\\ \tilde\alpha_i=\alpha_i \forall i\notin I}} W(\tilde\alpha) - W(\alpha)\,,$$

which expresses the gain in the dual objective function resulting from an optimization over the directions contained in I. To simplify notations, we write $\sigma(\alpha|i) := \sigma(\alpha|\{i\})$ and $\gamma(\alpha|i) := \gamma(\alpha|\{i\})$. Note that we have

$$\sigma(\alpha|i) = \sup_{\tilde\alpha_i\in[0,C_i]} (\tilde\alpha_i - \alpha_i)\nabla W_i(\alpha)\,,$$

while $\gamma(\alpha|i)$ expresses the gain

$$W\big(\alpha + (\alpha_i^{new} - \alpha_i)e_i\big) - W(\alpha)$$

of the 1D-update in direction i, where α_i^{new} is defined by (4). In addition, $\gamma(\alpha|\{i,j\})$ is the gain obtained by the update discussed in Section 3.1. Moreover, for $I = \{1,\ldots,n\}$ we write $\sigma(\alpha) := \sigma(\alpha|I)$ and $\gamma(\alpha) := \gamma(\alpha|I)$, respectively. Note that both σ and γ are monotonic in I, that is, for $I \subset J$ we have $\sigma(\alpha|I) \le \sigma(\alpha|J)$ and $\gamma(\alpha|I) \le \gamma(\alpha|J)$. Finally, we need the obvious relation

$$\gamma(\alpha) = W(\alpha^*) - W(\alpha)\,,$$

where we recall from Section 2 that $\alpha^* \in [0,C]$ denotes a solution of the dual problem (3). In other words, $\gamma(\alpha)$ expresses the dual sub-optimality of α.

Let us now begin our analysis by presenting two lemmata that establish relationships between these quantities.

Lemma 3 *For all* $\alpha \in [0,C]$ *we have*

$$\sum_{i=1}^n \sigma(\alpha|i) = \sigma(\alpha) = \mathrm{gap}(\alpha)\,,$$

where $\mathrm{gap}(\alpha)$ *denotes the duality gap defined in (7). In particular, there exists an index* $i^\star \in \{1,\ldots,n\}$ *such that*

$$\sigma(\alpha|i^\star) \ge n^{-1}\sigma(\alpha)\,.$$

This lemma can be easily derived from results in List et al. (2007) and List and Simon (2007). However, in the case of SVMs without offset, its proof is very elementary and hence we present it here for the sake of completeness.

Proof For $i \in \{1,\ldots,n\}$ it is easy to see that the supremum used to define $\sigma(\alpha|i)$ is attained at

$$\bar\alpha_i := \begin{cases} C_i & \text{if } \nabla W_i(\alpha) \ge 0 \\ 0 & \text{if } \nabla W_i(\alpha) < 0\,. \end{cases} \tag{13}$$

Moreover, the vector $\bar\alpha := (\bar\alpha_1,\ldots,\bar\alpha_n) \in [0,C]$ realizes the supremum defining $\sigma(\alpha)$, and hence we obtain

$$\sum_{i=1}^n \sigma(\alpha|i) = \sum_{i=1}^n \big\langle \nabla W(\alpha), (\bar\alpha_i - \alpha_i)e_i \big\rangle = \langle \nabla W(\alpha), \bar\alpha - \alpha\rangle = \sigma(\alpha)\,.$$

Furthermore, we have

$$\begin{aligned}\sigma(\alpha) = \langle \nabla W(\alpha), \bar{\alpha} - \alpha \rangle &= \langle \alpha, K\alpha \rangle - \langle e, \alpha \rangle + \sum_{i=1}^{n} \bar{\alpha}_i \cdot \nabla W_i(\alpha) \\ &= \langle \alpha, K\alpha \rangle - \langle e, \alpha \rangle + \sum_{i=1}^{n} C_i \left[\nabla W_i(\alpha)\right]_0^\infty ,\end{aligned}$$

and therefore we have shown $\sigma(\alpha) = \operatorname{gap}(\alpha)$. The last assertion is a trivial consequence of the first assertion. ■

The second lemma relates $\sigma(\alpha|I)$ to the gain $\gamma(\alpha|I)$. For its formulation we need the quantity $B_{\max} := \max_{i=1,\dots,n} C_i$.

Lemma 4 *For all* $\alpha \in [0,C]$ *and* $I \subset \{1,\dots,n\}$ *we have*

$$\sigma(\alpha|I) \;\geq\; \gamma(\alpha|I) \;\geq\; \frac{\sigma(\alpha|I)}{2} \min\left\{1, \frac{\sigma(\alpha|I)}{|I|^2 B_{\max}^2}\right\},$$

where $|I|$ *denotes the cardinality of* I.

In a slightly different form, this lemma has been established in, for example, Hush et al. (2006), and it was somewhat implicitly used in List and Simon (2007). Again, we present its proof for the sake of completeness.

Proof Let $\bar{\alpha}_i$ be defined by (13) and $d := \sum_{i \in I} (\bar{\alpha}_i - \alpha_i) e_i$. For $\lambda \in [0,1]$, we then have $\alpha + \lambda d \in [0,C]$, and a calculation analogous to the one in the proof of Lemma 1 yields

$$\gamma(\alpha|I) \geq W(\alpha + \lambda d) - W(\alpha) = \lambda \big\langle \nabla W(\alpha), d \big\rangle - \frac{\lambda^2}{2} \langle d, Kd \rangle \geq \lambda \sigma(\alpha|I) - \frac{\lambda^2 |I|^2 B_{\max}^2}{2} .$$

Now the right hand side is maximized at

$$\lambda^* := \begin{cases} 1 & \text{if } \sigma(\alpha|I) > |I|^2 B_{\max}^2 \\ |I|^{-2} B_{\max}^{-2} \sigma(\alpha|I) & \text{if } \sigma(\alpha|I) \leq |I|^2 B_{\max}^2 . \end{cases}$$

In the case $\sigma(\alpha|I) > |I|^2 B_{\max}^2$ we hence find

$$\gamma(\alpha|I) \geq \sigma(\alpha|I) - \frac{|I|^2 B_{\max}^2}{2} > \frac{\sigma(\alpha|I)}{2} ,$$

while in the other case $\sigma(\alpha|I) \leq |I|^2 B_{\max}^2$ we obtain

$$\gamma(\alpha|I) \geq \frac{\sigma^2(\alpha|I)}{2|I|^2 B_{\max}^2} .$$

Combining all estimates we then obtain the inequality on the right hand side.

To show the inequality on the left hand side we fix an $\tilde{\alpha} \in [0,C]$ such that $\tilde{\alpha}_i = \alpha_i$ for all $i \notin I$. Then we have

$$W(\tilde{\alpha}) - W(\alpha) = \langle \nabla W(\alpha), \tilde{\alpha} - \alpha \rangle - \frac{1}{2} \langle \tilde{\alpha} - \alpha, K(\tilde{\alpha} - \alpha) \rangle \leq \langle \nabla W(\alpha), \tilde{\alpha} - \alpha \rangle \leq \sigma(\alpha|I) ,$$

and by maximizing the left hand side of this inequality over $\tilde{\alpha}$ we find $\gamma(\alpha|I) \leq \sigma(\alpha|I)$. ■

With these preparations we can now present a preliminary result on iterative algorithms that have a certain control of their gain.

Proposition 5 *Let $\alpha^{(0)}, \alpha^{(1)}, \dots \in [0,C]$ be a sequence of feasible vectors that satisfies*

$$W(\alpha^{(\ell+1)}) - W(\alpha^{(\ell)}) \geq \gamma(\alpha^{(\ell)}|i^\star_\ell), \qquad \ell \geq 0, \tag{14}$$

where for each ℓ the index $i^\star_\ell \in \{1,\dots,n\}$ is the one described in Lemma 3, that is, it satisfies $\sigma(\alpha^{(\ell)}|i^\star_\ell) \geq n^{-1}\sigma(\alpha^{(\ell)})$. Then for all $\ell \geq 1$ we have

$$\gamma(\alpha^{(\ell+1)}) \leq \gamma(\alpha^{(\ell)}) \left(1 - \frac{1}{2n} \min\left\{1, \frac{\gamma(\alpha^{(\ell)})}{nB^2_{\max}}\right\}\right).$$

Moreover, for all $\varepsilon > 0$ and all $\ell \geq \ell_\varepsilon$ we have $\gamma(\alpha^{(\ell)}) \leq \varepsilon$, where

$$\ell_\varepsilon := \left\lceil \frac{2n^2 B^2_{\max}}{\varepsilon} \right\rceil + \max\left\{0, \left\lceil 2n \ln \frac{W(\alpha^*) - W(\alpha^{(0)})}{\varepsilon} \right\rceil \right\}.$$

Proof By Lemmas 4 and 3 we find

$$\begin{aligned} \gamma(\alpha^{(\ell)}) - \gamma(\alpha^{(\ell+1)}) = W(\alpha^{(\ell+1)}) - W(\alpha^{(\ell)}) &\geq \gamma(\alpha^{(\ell)}|i^\star_\ell) \\ &\geq \frac{\sigma(\alpha^{(\ell)}|i^\star_\ell)}{2} \min\left\{1, \frac{\sigma(\alpha^{(\ell)}|i^\star_\ell)}{B^2_{\max}}\right\} \\ &\geq \frac{\sigma(\alpha^{(\ell)})}{2n} \min\left\{1, \frac{\sigma(\alpha^{(\ell)})}{nB^2_{\max}}\right\} \\ &\geq \frac{\gamma(\alpha^{(\ell)})}{2n} \min\left\{1, \frac{\gamma(\alpha^{(\ell)})}{nB^2_{\max}}\right\}. \end{aligned}$$

From this we easily obtain the first assertion.

The second assertion has already been shown in the second part of the proof of the first assertion of (List and Simon, 2007, Theorem 4), which can be found on the pages 312 and 313 of List and Simon (2007). ■

Note that $1/n$-rate certifying algorithms considered in List and Simon (2007) clearly satisfy assumption (14). Moreover, Proposition 5 can also be applied to the 1D-SVM and 2D-SVM:

Theorem 6 *Consider the* 1D-SVM *described in Algorithm 1 or a* 2D-SVM *in the sense of Algorithm 2 that uses a working set selection strategy whose gain at each iteration is not less than that of the* 1D-SVM. *Furthermore, assume that $\max\{w_{\text{neg}}, w_{\text{pos}}\} \leq 1$. Then for all $\varepsilon > 0$, $n \geq 1$, and all $\lambda > 0$ these algorithms terminate after at most*

$$\left\lceil \frac{2}{\lambda\varepsilon \min\{1, 2\lambda\varepsilon\}} \right\rceil + \max\left\{0, \left\lceil 2n \ln \frac{4\lambda(W(\alpha^*) - W(\alpha^{(0)}))}{\varepsilon \min\{1, 2\lambda\varepsilon\}} \right\rceil \right\}$$

iterations. In particular, in the most likely scenario $2\lambda\varepsilon \leq 1$ *these algorithms do not need more iterations than*

$$\left\lceil \frac{1}{\lambda^2\varepsilon^2} \right\rceil + \max\left\{0, \left\lceil 2n \ln \frac{2(W(\alpha^*) - W(\alpha^{(0)}))}{\varepsilon^2} \right\rceil \right\}.$$

Proof The 1D-SVM chooses at each iteration ℓ a direction i^*_ℓ that maximizes the 1D-gain $\gamma(\alpha^{(\ell)}|i)$. Consequently, we have

$$W(\alpha^{(\ell+1)}) - W(\alpha^{(\ell)}) = \gamma(\alpha^{(\ell)}|i^*_\ell) \geq \gamma(\alpha^{(\ell)}|i^\star_\ell),$$

where $i^\star_\ell$ is the direction described in Lemma 3. In other words, (14) is satisfied for this algorithm, and from this it is not hard to see that the considered 2D-SVM's also satisfy assumption (14). Let us now define

$$h(\sigma) := \frac{\sigma}{2} \min\left\{1, \frac{\sigma}{n^2 B^2_{\max}}\right\}, \qquad \sigma > 0.$$

For $\varepsilon := h(\frac{\varepsilon}{2\lambda})$ Proposition 5 together with Lemma 4 then shows that

$$h\Big(\frac{\varepsilon}{2\lambda}\Big) = \varepsilon \geq \gamma(\alpha^{(\ell)}) \geq h\big(\sigma(\alpha^{(\ell)})\big)$$

for all $\ell \geq \ell_\varepsilon$ and hence we obtain $S(\alpha^{(\ell)}) \leq \text{gap}(\alpha^{(\ell)}) = \sigma(\alpha^{(\ell)}) \leq \frac{\varepsilon}{2\lambda}$ by the monotonicity of the function h. Using $B_{\max} \leq \frac{1}{2\lambda n}$ we then obtain the assertion by simple algebraic transformations. ■

Note that the working set selection strategies WSS 1, WSS 2, WSS 4, WSS 8, WSS 32, WSS 64, WSS 128, WSS 512, and WSS 1024, satisfy the assumptions of Theorem 6. Moreover, the same is true for all combinations of working set selection strategies that include at least one of the strategies listed. Finally, note that the upper bound established in Proposition 5 coincide (modulo constants that come from different working set sizes) with the bounds for rate certifying algorithms presented in List and Simon (2005), Hush et al. (2006) and List and Simon (2007). Moreover, the step from dual ε-optimality to primal ε-optimality considered in the proof of Theorem 6 coincides with the analysis (List et al., 2007) for SVMs with offset. Consequently, the bound presented in Theorem 6 equals the best known guarantees for solvers for SVMs with offset.

5. Experiments

The described 1D-SVM-solver and 2D-SVM-solver enjoy nice theoretical properties with respect to both generalization performance and required training time. However, it is unclear how tight these bounds are, so it remains unclear whether the proposed SVMs also perform well in practice. Therefore, we performed several experiments that address the following questions:

1. Which subset selection strategies lead to the smallest number of iterations or the shortest run time? How many more iterations than WSS 0 do these strategies perform?

2. How many less iterations needs the stopping criterion (9) compared to standard duality gap (7) and is there also an advantage in terms of run time?

3. How much more efficient is the 2D-SVM-solver than the technically much easier 1D-SVM-solver?

4. How well does the 2D-SVM-solver work compared to standard software packages such as LIBSVM by Chang and Lin (2009)?

5. What is the advantage of warm start initializations when the parameter search is performed over a grid?

To answer these questions we implemented the 1D-SVM- and the 2D-SVM-solver in C++, and downloaded LIBSVM version 2.82 by Chang and Lin (2009). The algorithms were compiled by LINUX's gcc version 4.3 with various software and hardware optimizations enabled. All experiments were conducted on a computer with INTEL XEON X5355 (2.66 GHz) quad core processor and 8GB RAM under a 64bit version of RedHat Linux Enterprise 4. During all experiments that incorporated measurements of run time, one core was used solely for the experiments, and the number of other processes running on the system was minimized. The run time itself was measured by the C function clock() from the library time.h. The resulting resolution was 0.01 seconds.

In some preliminary experiments we made a couple of observations that changed the described implementation strategy slightly: First, it turned out that the auto-vectorization of gcc only gave mediocre and sometimes even contradicting results, even if the implementation guidelines of gcc 4.3's auto-vectorization were strictly followed. Therefore, we decided to manually code SSE2-vectorized versions of the most important routines, namely: computing kernel values, searching for the optimal 1D-direction, updating the gradient, and computing the weighted sum $E(\alpha)$ of clipped slack variables. To this end, we used the library emmintrin.h together with properly aligned arrays of doubles.[4] Some of our preliminary experiments not reported here indicated that this specialized hardware instruction set yields a run time improvement by a factor between 1.3 and 1.8 depending on the working set selection strategy and the data set. Second, the initial experiments suggested substantial numerical instabilities on a few data sets when using single floats, so we decided to use double precision throughout the experiments. Third, we were rather disappointed by the run time behavior of LIBSVM, even when we enabled its shrinking heuristic.[5] After some investigations we found that the main reason for the disappointing run time performance was the fact that LIBSVM copies kernel rows into the kernel cache, if one uses pre-computed kernel matrices, which, as discussed below, we did throughout the experiments. This copying mechanism results in a small number of iterations per second when the LIBSVM-solver is started on a new parameter point, while with the kernel cache being filled up during the optimization, the solver starts performing more iterations per second. To ensure a fair comparison, we thus decided to implement our own version of LIBSVM's solver (without shrinking strategy). As a side effect, this new implementation also benefited from the SSE2 instructions for upgrading the gradient. Unlike the subset selection strategy of the 1D-SVM-solver, however, LIBSVM's subset selection strategy, though implementable, does

[4] At first glance, this manual approach may seem to be too specialized, since it should clearly be not the goal of this paper to fine-tune an algorithm to a very specific hardware environment. On the other hand, a good compiler should make optimizations with respect to these nowadays standard instructions, which have been first introduced by Intel in 2001 and have been adopted by AMD in 2003, automatically. Unfortunately, it turned out that gcc 4.3 did not do this optimization reliably. Namely, depending on some minor and apparently independent changes in other parts of the code, the most crucial loops where sometimes optimized and sometimes not. This behavior rendered a reasonable comparison of different algorithms impossible. Therefore, our manual approach can also be viewed as a compilation with a more ideal compiler, which in the future is hopefully available.

[5] In fact, it turned out that neither the number of iterations nor the run time was significantly affected by the shrinking heuristic. Corresponding results for the run time are reported in Figure 1.

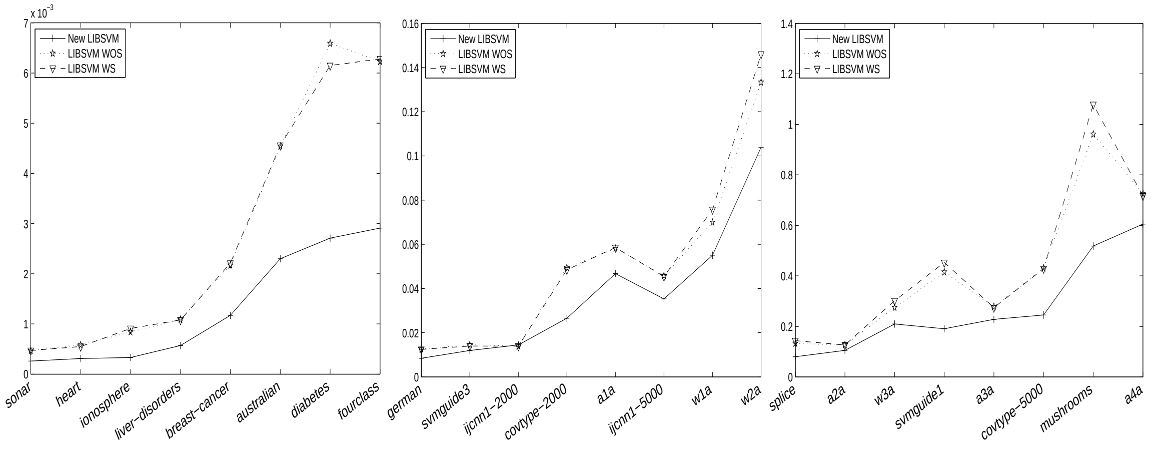

Figure 1: Performance of the original LIBSVM-solver (WOS: without shrinking; WS: with shrinking) compared to our own implementation of the LIBSVM-solver. The graphic displays the average run time in seconds (middle) over the 10 by 10 parameter grid described later in this section. Shrinking does not give an advantage on *this* parameter grid, while the new LIBSVM implementation runs in almost all cases significantly faster than the original LIBSVM .

not benefit from vectorization since not all indices are considered, and hence the relatively slow non-serial RAM access of the CPU outweighs the speed improvement of the SSE2 instructions.

We downloaded all data sets for binary classification from LIBSVM's homepage whose number of features did not exceed 1000. We made this cut because having data sets with a huge number of features would have required substantial extra effort for implementing our algorithms, and this effort was clearly out of the scope of this paper. In all cases, we used the scaled versions of these data sets, and if they were not available, we scaled the unscaled data sets with the help of LIBSVM's scaling tool. For data sets that were not split into a training and test set we generated a random split that contained approximately 70% training and 30% test samples. Moreover, for the already split data sets SPLICE, SVMGUIDE1, SVMGUIDE3, we decided to first merge the corresponding training and test set and then generate the random split above. For the large data sets COVTYPE and IJCNN1, we generated random subsets of the two data sets of sizes $n = 2000, 5000$, and then applied the random split above. Finally, we ignored some versions with larger training set of the AXA and WXA families, namely A5A–A9A, and W4A–W8A because of time and memory constraints. Moreover, for these two families of data sets we kept the split between training and test sets. Table 1 shows the corresponding characteristics of the considered data sets together with classification errors of the fastest version of the 2D-SVM and LIBSVM, respectively.

In all our experiments, we considered k-fold cross validation with folds randomly generated from the training set and hyper-parameters λ and σ each taken from a 10 by 10 grid. Since the choice of this grid has a significant influence on both the training time and the learning performance, special care is needed here. Despite such care, however, it seems likely that every choice will be subject to discussion. To pick the parameter grid less heuristically than in previous investigations, we decided to use recent statistical insights from Steinwart et al. (2007), which show that *asymptotically* good

	training size	test size	dimension	LIBSVM	2D-SVM	2D-SVM (duality gap)	2D-SVM (fine grid)
SONAR	146	62	60	12.68 ± 4.27	12.80 ± 4.04	12.62 ± 3.95	13.21 ± 4.19
HEART	188	82	13	17.58 ± 3.80	17.42 ± 4.39	17.47 ± 3.86	17.94 ± 4.21
LIVER-DISORDERS	248	97	6	29.31 ± 4.00	29.76 ± 4.31	28.90 ± 3.94	28.70 ± 4.25
IONOSPHERE	248	103	34	5.43 ± 2.16	8.59 ± 2.85	8.33 ± 2.78	8.76 ± 2.97
AUSTRALIAN	484	206	14	14.76 ± 2.21	14.54 ± 2.09	14.77 ± 1.90	14.45 ± 2.24
BREAST-CANCER	493	190	10	3.30 ± 1.06	3.15 ± 1.07	3.13 ± 1.01	3.15 ± 1.02
DIABETES	544	334	8	23.43 ± 2.38	23.68 ± 2.49	23.62 ± 2.33	23.80 ± 2.41
FOURCLASS	623	239	2	0.09 ± 0.18	0.04 ± 0.14	0.04 ± 0.14	0.08 ± 0.18
GERMAN.NUMER	718	282	24	24.95 ± 2.27	24.84 ± 2.29	24.93 ± 2.16	25.35 ± 2.10
SVMGUIDE3	892	392	21	16.48 ± 1.70	16.60 ± 1.77	16.55 ± 1.74	16.42 ± 1.68
COVTYPE-2000	1392	616	54	24.06 ± 1.60	23.92 ± 1.69	24.09 ± 1.49	24.14 ± 1.57
IJCNN1-2000	1424	584	33	4.38 ± 0.91	4.38 ± 0.93	4.37 ± 0.96	4.36 ± 0.86
A1A	1605	30956	123	15.78 ± 0.17	15.89 ± 0.21	15.75 ± 0.14	16.11 ± 0.51
SPLICE	2176	999	60	8.68 ± 0.87	8.93 ± 0.88	8.89 ± 0.90	8.79 ± 0.94
A2A	3365	30296	123	15.76 ± 0.30	15.74 ± 0.27	15.76 ± 0.28	15.97 ± 0.44
W1A	2477	47332	300	2.20 ± 0.07	2.18 ± 0.06	2.21 ± 0.07	2.22 ± 0.07
A3A	3185	29336	123	15.57 ± 0.08	15.82 ± 0.21	15.55 ± 0.10	15.85 ± 0.22
W2A	3470	46339	300	1.94 ± 0.09	1.95 ± 0.06	1.97 ± 0.06	1.94 ± 0.06
COVTYPE-5000	3472	1536	54	20.77 ± 0.88	20.73 ± 0.83	20.74 ± 0.86	20.79 ± 0.81
IJCNN1-5000	3486	1514	33	2.73 ± 0.42	2.70 ± 0.45	2.72 ± 0.41	2.70 ± 0.45
A4A	4781	33780	123	15.52 ± 0.07	15.80 ± 0.30	15.58 ± 0.13	15.64 ± 0.14
W3A	4912	44833	300	1.75 ± 0.05	1.75 ± 0.05	1.76 ± 0.05	1.72 ± 0.06
SVMGUIDE1	4959	2130	4	2.97 ± 0.32	3.01 ± 0.33	2.97 ± 0.32	3.03 ± 0.29
MUSHROOMS	5773	2351	112	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.01

Table 1: Characteristics of the considered data sets together with the test errors (± standard deviations) on 100 random splits. The training and test set sizes refer to the splits used in the run time experiments. The considered algorithms were the 2D-SVM with WSS 7, I1-W4 and clipped duality gap stopping criterion (6th column), LIBSVM (5th column), the 2D-SVM with WSS 7, I1-W4 and duality gap stopping criterion (7th column), and another 2D-SVM with WSS 7, I1-W4 and clipped duality gap stopping criterion (8th column). The hyper-parameters for the first three test error columns were selected by 10-fold cross-validation on the 10 by 10 grid described in the text, while for the fourth test error column, 10-fold cross-validation on a finer and larger 25 by 30 grid was used.

values of λ and σ are contained in the intervals $[c_1 n^{-2}, 1]$ and $[c_2, c_3 n^{1/d}]$, respectively, where n is the number of training samples, d is the input dimension, and c_1, c_2, and c_3 are arbitrarily specifiable constants independent of n and d. Based on this result, we considered a geometrically spaced 10 by 10 grid in $[10n^{-2}, 1] \times [0.1, 2n^{1/d}]$, that is, the ratio of consecutive grid points was constant. Here, it is worth mentioning that during the k-fold cross validation λ was internally converted to C by the formula $C := \frac{k}{2(k-1)\lambda n}$ to accommodate the fact that the *actual* training set size for k-fold cross validation is approximately $(k-1)n/k$. To empirically validate the quality of this grid with respect to classification performance we repeated the 10-fold cross validation procedure 100 times for the fastest versions of the 2D-SVM and LIBSVM, respectively. We refer to Section 5.1 for an exact description of the experimental setup. The resulting average classification errors, which are reported in Table 1, show that both algorithms achieve comparable classification performance except on one small data set, namely IONOSPHERE. However, the size of this and some other data sets make it hard to draw conclusion from the corresponding, reported errors. While this experiment showed, that both algorithms performed equally well on the chosen grid, it does not allow statements about the absolute quality of the grid. We therefore conducted a control experiment[6] with the fastest version of the 2D-SVM on a geometrically spaced 25 by 30 grid in $[0.001n^{-2}, 1] \times [0.005, 20n^{1/d}]$. The size and boundaries of this control grid ensured that it was both significantly finer and larger than the 10 by 10 grid. Besides the different grid size, the experimental setup followed that described in Section 5.1 and the resulting average classification errors are reported in Table 1, too. The results in this table show that the classification performance is not improved when using the larger grid, which in turn means that our initial 10 by 10 grid is a good choice.

Like the choice of the grid, the stopping criterion and its threshold value have a significant influence of the number of iterations and the run time of an SVM solver. Unfortunately, the 2D-SVM and LIBSVM use different stopping criteria, which makes a direct comparison difficult. To address this problem, we again took a statistical perspective in the sense that the ultimate goal when solving the SVM optimization problem is not numerical but statistical accuracy. In other words, we may stop the iterative optimization procedures as soon as we know that the remaining inaccuracy does not significantly influence the classification performance. For the 1D-SVM and the 2D-SVM we thus used the stopping criterion (9) with $\varepsilon := 0.001$, while for our version of LIBSVM's solver we used, like the original LIBSVM, the classical MVP stopping criterion with $\varepsilon = 0.001$. Here we note that this was necessary since LIBSVM's solver deals with SVMs *with* offset b, and hence the stopping criterion (9) is no longer applicable. In addition, an appropriately modified stopping criterion seems to be computationally inefficient, while by (List et al., 2007, Lemma 8) the MVP stopping criterion with value $\varepsilon = 0.001$ also ensures (8) for $\varepsilon := 0.001$ and f^* instead of $[f^*]_{-1}^{1}$. In other words, LIBSVM's default value, which we picked throughout our experiments, actually has a good interpretation in terms of learning. Of course, the different stopping criteria used raise the question whether the results reported below are due to differences in the working set selection strategy, the different nature of the optimization problem, *or* the stopping criteria. In this regard, we note that in the experiments with LIBSVM our goal is to compare the *entire* 2D-SVM-solver with a state-of-the-art solver, rather than to, for example, compare different working set selection strategies. For this purpose, it is irrelevant whether the working set selection strategy, the nature of the optimization problem, or the different stopping criteria have a stronger influence on the run

[6] This control experiment was extremely time-consuming, and hence we were forced to distribute the runs between different machines (with different hardware features). For the same reason, it was, unfortunately, infeasible to run the same control experiment for LIBSVM.

time. Nonetheless, it remains an interesting question for future work whether solver's for SVMs with offset can also benefit from some of the ideas of the working set selection strategies introduced for the 2D-SVM-solver.

In all experiments we pre-computed the kernel matrix in order to avoid that these solver-independent but data set-dependent computations are contained in the reported training times. To be more precise on the latter dependence, recall that the time needed to compute the matrix K significantly depends on the number of features of the samples and the implementation-specific internal representation of the samples. For example, we may have two data sets in $\mathbb{R}^{d_1}$ and $\mathbb{R}^{d_2}$, respectively, that generate the same matrix K. Now assume that $d_1 \ll d_2$. Without pre-computing the kernel matrix, the solver will need significantly more time for the second data set, while the run times for both data sets will be equal for pre-computed K. Moreover, the second data set may actually consist of samples for which most of the coordinates are zero. In this case, an internal data representation like LIBSVM's that exploits this sparseness may speed up the computation of both the entire K and single kernel matrix rows. On the other hand, if the data does *not* enjoy this kind of sparseness, a straightforward sample representation by arrays is typically superior, since it avoids costly branches, allows sequential RAM access, which, from our experience, is often 4 times faster than random access, and makes it possible to use vectorization features of modern processors such as SSE2 instructions. Last but not least, we observed recently, after the experiments of this paper were finished, that the pre-computation of K enjoys an almost linear speed-up, when it is distributed among the cores of modern multi-core processors, while for the computation of single rows of K the improvement may be significantly less due to the time spend for synchronization.[7] Obviously, these implementation options make it impossible to determine a canonical method for dealing with the kernel matrix K, whether it fits into memory or not. Consequently, by picking a particular method and including its run time into the measurements, one necessarily introduces a bias into the experiments, and hence run time results reported from a series of such experiments may be of little value for new, time-critical SVM applications with different data set characteristics.[8] In fact, for such applications all the considerations above need to be carefully taken into account, which in turn requires knowledge of the computational complexity of each *individual* component. In other words, for an informed decision one needs to know, among others, the run time complexity of the core solver, which in turn gives another argument for considering the core solver with pre-computed matrix.[9] On the downside, however, this approach is, of course, unrealistic for large data sets whose kernel matrices do not fit into the computers memory. On the other hand, for all considered data sets the matrices *did* fit into memory, and in addition, it turned out that for all data sets there were parameter regions of the grid where all or basically all vectors were support vectors. In these regions, the corresponding kernel rows would have been computed during the optimization,

[7] This observation suggests that in the future, the computation of K, which is currently a significant part of the entire SVM training time, may be significantly less time consuming. This may be in particular true for highly parallel architectures such as graphical processing units.

[8] In an extreme case, including the computation time for K opens the possibility that a new solver appears to be faster than existing ones simply because it has a better implementation for computing K.

[9] Another approach would be to *a)* not pre-compute K and *b)* exclude the time needed to compute and cache kernel rows from the reported run time. Unfortunately, this approach is infeasible because of the relatively low resolution of the time-measuring functions in C.

Yet another approach would be to actually precompute K, but to pretend that a cache of a certain size is used. One could then log cache misses of such a virtual cache. While the latter approach may actually be the silver bullet for future work, the idea for it only arose after the reviewers comments, and, unfortunately, a respective re-design of our experiments were too costly at that stage.

if we had not precomputed the kernel matrices, and consequently, training over the grid would have required the solver to compute the kernel matrix anyway. From this point of view, our experiments suggests that training over a grid with medium-sized data sets whose kernel matrix still fits into memory, there is no need to implement a caching strategy. In fact, we strongly conjecture that without pre-computing the kernel matrices, our experiments would have rendered computationally infeasible with one computer only. It is, of course, needless to say, that the situation may change, if other parameter selection strategies are used, or the data sets are too large.

5.1 Comparing Classification Performance

When comparing the standard SVM optimization problem with the version in (1), probably the first question is, whether the absence of the offset term has an influence on the classification performance. To answer this question we performed on each data set 100 runs for both a version of the 2D-SVM-solver and our implementation of LIBSVM's solver. We performed these experiments, though we report them first, actually at the very end of our investigations. This way, we could use for each solver the fastest version. For the 2D-SVM-solver it turned out, as we will see below, that this is the WSS 7 strategy together with I1-W4 initialization, while for the LIBSVM's solver we used, depending on the data set, either I1-W2 or I1-W5 initialization. Besides for the data sets of the AXA and WXA families, we generated for each data set 100 random splits, where each training set contained, modulo randomness, 70% of the samples. Moreover, on each of these training sets the hyper-parameter selection was performed by 10 fold cross-validation over the parameter grid described above. The test error was then computed on the test set part of the split, which, modulo randomness, contained 30% of the samples. The resulting average classification errors are reported in Table 1. As one quickly observes, LIBSVM yielded the better classification performance on the data set IONOSPHERE. On all other data sets, however, both algorithms performed almost indistinguishable. Therefore, it seems fair to conclude that the classification performance is not significantly influenced by the absence of the offset.

5.2 Comparisons to the 2D Selection Strategy with Maximal Gain

In our first set of experiments on 2D subset selection strategies, we investigated the number of iterations needed for the different strategies for selecting working sets. Our baselines were the 1D-SVM-solver and the 2D-SVM strategy WSS 0, which searches for the pair of indices with maximal dual gain. Since the latter is computationally very expensive, we decided to use only 2-fold cross validation. In addition, we actually trained only on one of the two folds, that is, our approach is best described by the training/validation SVM (TV-SVM) of Steinwart and Christmann (2008). Besides these modifications, however, we followed the approach outlined earlier. Finally, in all experiments of this subsection, we initialized by $\alpha \leftarrow 0$.

Let us now have a closer look at the results that are displayed in Figures 2 to 5. Figure 2 compares the 1D-SVM, WSS 0 and the simple 2D-modifications of the 1D-SVM. Not surprisingly, WSS 0 needs substantially less iterations than its one-dimensional equivalent 1D-SVM, while all of the simple 2D-modifications perform somewhere in between. More precisely, WSS 1 yields some significant improvement over the 1D-SVM. For WSS 2 the message is mixed; while on some data sets, WSS 2 performs significantly better, the difference is more marginal on other data sets. However, combining WSS 1 and WSS 2 into WSS 3 yields a clear overall improvement over both methods and the 1D-SVM. Another combination, WSS 5 that combines WSS 1 with a search over

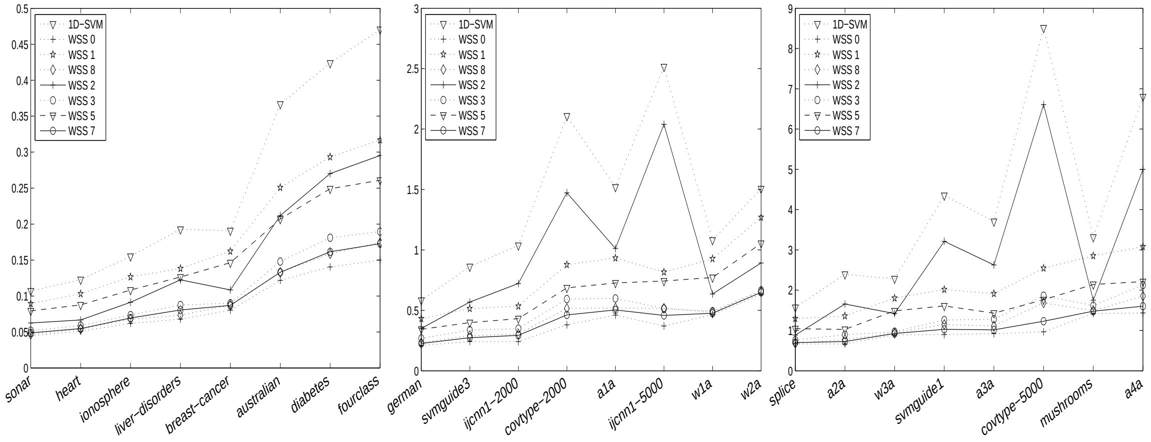

Figure 2: Performance of methods based on simple extensions of the 1D-search strategy for small (left), mid-sized (middle), and relatively large data sets (right). The graphic displays the average number of iterations in thousands for the different methods over the entire 10 by 10 parameter grid. All 2D-methods perform better than the 1D-SVM, but the degree of improvement differs significantly. WSS 2 performs sometimes better and sometimes substantially worse than WSS 1, but combining both methods into WSS 3 leads to uniform improvements. The same holds for WSS 5, though with less improvements. The combination WSS 7 uniformly yields the lowest number of iterations.

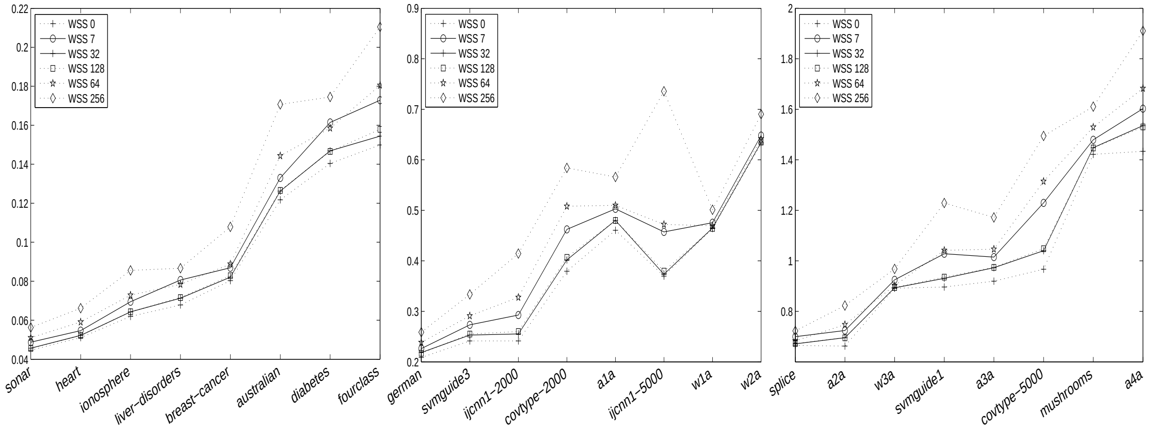

Figure 3: Performance of methods based on approximations of the 2D strategy WSS 0 (black). The graphic displays the average number of iterations in thousands for the different methods over the entire 10 by 10 parameter grid. WSS 0 performs uniformly best, but both deterministic strategies WSS 32 and WSS 128, which are basically indistinguishable, closely follow the performance of WSS 0. WSS 7 and the hybrid WSS 64 still capture most of the behavior of the previous methods with small advantages for WSS 7, while the complete randomization performs uniformly worst.

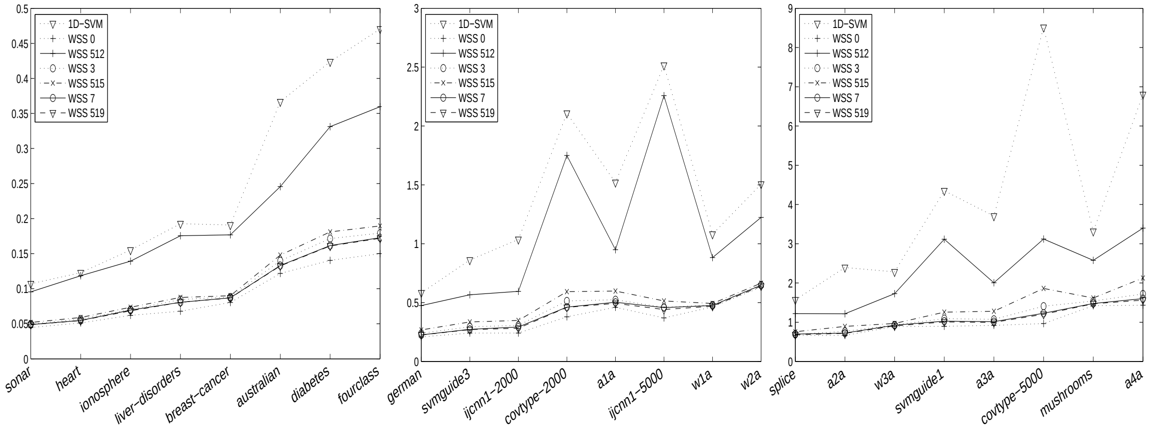

Figure 4: Combining methods based on simple 1D-extensions with WSS 512, which considers the approximate gain on inner SVs. The graphic displays the average number of iterations in thousands for the different methods over the entire 10 by 10 parameter grid. Without combining WSS 512 with other methods, it performs quite poorly, while combining WSS 512 with WSS 3 to WSS 515 yields an improvement over both methods. In contrast, combining WSS 512 and WSS 7 to WSS 519 does not give an improvement over WSS 7 as the almost indistinguishable two green lines show.

10 nearest neighbors, also needs substantially less iterations than WSS 1 and the 1D-SVM, but the improvements are less than those of WSS 3. However, the combination of all, WSS 7, does not only perform uniformly better than all participating methods, but also needs in most cases only a few more iterations than the optimal WSS 0. Finally, WSS 8, which is a variant of WSS 1, also reduces the number of iterations substantially, yet it fails to perform as well as WSS 7. Let us now have a closer look at Figure 3 that shows how the methods based on an approximation of the optimal WSS 0 perform. Here it turns that WSS 32, which uses the exact computation of the 2D-gain, and WSS 128, which uses an approximation of the 2D-gain, perform indistinguishably. In addition, they only need a few more iterations than WSS 0, and constantly outperform WSS 7, yet the latter improvement is in most cases only marginal. Finally, the random approaches WSS 64 and WSS 256 do not need less iterations than WSS 7, and the complete random approach of WSS 256 performs worse than the hybrid strategy WSS 64. However, by comparing with Figure 2 we see that WSS 256 still needs significantly less iterations than the 1D-SVM.

Another way to approximately compute the 2D-gain is implemented in WSS 512. Figure 4 compares the number of iterations of this method to the 1D-SVM, WSS 0, and some combinations of WSS 512 with simple 2D-extensions of the 1D-SVM approach. A closer look at this figure shows that WSS 512 alone is not a very good alternative to the 1D-SVM, while combinations do yield significant improvement. However, these improvements are not significantly better than WSS 7.

Finally, let us compare the 1D-SVM and the optimal 2D strategy WSS 2 with the MVP approach of WSS 16 and LIBSVM. Figure 5 shows that the 2D-MVP approach of WSS 16 performs only slightly better than the 1D-SVM. In contrast, LIBSVM needs, not surprisingly, substantially less iterations than the 1D-SVM, but it fails to perform as well as the simple WSS 3, and the more complicated WSS 7.

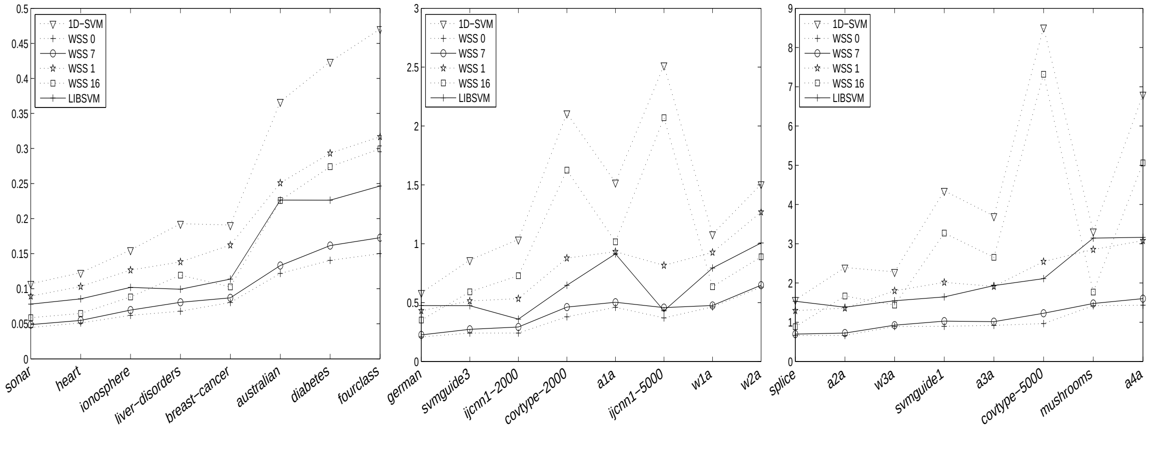

Figure 5: LIBSVM and MVP compared to some other approaches. The graphic displays the average number of iterations in thousands for the different methods over the entire 10 by 10 parameter grid. On all data sets considered, the 2D-MVP strategy WSS 16 has some advantage over the simple 1D-SVM, while LIBSVM often needs substantially less iterations and performs comparably to the WSS 1. However, neither of the methods approach the close-to-optimal performance of WSS 7 or even the optimal performance of WSS 0.

5.3 Comparisons of Different 2D Subset Selection Strategies

The experiments of the previous subsection identified some working set selection strategies that performed close to WSS 0 in terms of iterations. Unlike WSS 0, all these strategies were $O(n)$, yet is seems obvious, that their run time may substantially differ. Therefore, the goal of the experiments in this subsection is to evaluate the working set selection strategies in terms of their run time. To this end, we performed the already described 10-fold cross validation training on our data sets and measured both the number of iterations and the run time. Note that by considering both quantities simultaneously, it is possible to decide whether a strategy suffers from its large number of iterations or only from its computational requirements for selecting the working set. In the following, we only summarize our findings, since Appendix A.1 contains various graphics displaying our results of this subsection in detail. In this appendix, we always report the average requirements per grid point, where the average is either taken with respect to all 10 folds and the entire grid, or just with respect to the 10 folds and the grid points whose validation error is close to the minimal validation error. The latter averages are of particular interest, when one does not use grid search for the hyper-parameter selection, but some other, possibly faster methods, such as the one by Keerthi et al. (2007). In addition, the latter averages are also interesting for grid search, since after such a search one usually retrains the SVM on the entire training set with the hyper-parameters that performed best in terms of validation error.

Let us now have a closer look at the results. The first observation, see Figures 8 and 9 for details, is that WSS 2, which needs less iterations than the 1D-SVM, does not run substantially faster. However, this behavior can be relatively easily explained by the fact that in each iteration the 1D-SVM updates the gradients for one direction only, whereas WSS 2, due to its 2D-nature, performs two such updates per iteration. Similarly, WSS 8 cannot translate its advantage over WSS 1 in terms of iterations into a substantial advantage in terms of run time. In this case, a closer look reveals that, compared to WSS 1, WSS 8 performs an additional, implicit gradient upgrade when looking for the second direction j. The other results displayed in Figures 8 and 9 confirm our results from Figure 2. In particular, WSS 7 not only need the fewest number of iterations, but also runs fastest on almost all data sets. Finally, Figure 10 reveals, for which hyper-parameters some combined methods achieve their speed-up compared to WSS 1. In particular, for large values of λ, WSS 3 and WSS 7 need only half of the iterations of WSS 1 and WSS 5, which indicates that in this regime, WSS 2 is the dominating strategy in the former two combinations. On the other hand, for small values of λ, the nearest neighbor strategy WSS 4 seems to be the dominating working set selection strategy of WSS 5 and WSS 7, since both methods need substantially less iterations than the methods WSS 1 and WSS 3, which do not include the nearest neighbor strategy. Finally, these advantages in terms of iterations do translate into almost the same advantage in terms of run time, since the additional costs of the nearest neighbor strategy only depend on the number k of considered nearest neighbors, which, in general, is quite small compared to the sample size. Nonetheless it is worth mentioning that for a few hyper-parameter pairs, it is faster not to use the nearest neighbor strategy.

Let us now turn to the methods that try to approximate the working set strategy of the optimal WSS 0. Here, see Figures 11 and 12 for the details, it turns out, that WSS 32 and WSS 128, whose required number of iterations were closest to WSS 0, have a significant higher run time than WSS 7. Since the number of iterations of these three methods behave quite similarly, the only explanation for this different run time behavior is the additional cost per iteration for computing all (approximate) 2D-gains. This explanation is further confirmed by the fact that WSS 128, which involves the cheaper approximate 2D-gain has a better run time behavior than WSS 32, which uses the exact computation of the 2D-gain. Furthermore, WSS 64, which computes only a fifth of the 2D-gains WSS 32 computes, runs substantially faster than WSS 32, despite the fact the the former needs more iterations. In this regard, we finally note that WSS 256 runs over-proportionally slowly compared to, for example, WSS 128. Most likely, this behavior can be explained by less effective hardware caching for the random pair selection of WSS 256. To get a better impression, on how effective WSS 7 chooses its working sets, let us now have a closer look at the number of iterations of the different working set selection strategies. The bottom graphics of Figure 11 show that over the entire grid, WSS 7 only needs 5% to 20% more iterations than the best performing WSS 32. However, if one considers only the grid points with small validation error, this good behavior becomes worse. Indeed, the bottom graphics of Figure 12 show that for such hyper-parameters, WSS 7 typically needs more than 20% more iterations than WSS 32, and in some cases even more than 50% more. Finally, Figure 13 reveals that in particular for small values of λ and flat kernels, WSS 7 requires substantially more iterations than WSS 32. However, at least on the data set SVMGUIDE1 this worse behavior takes place at grid points that do not need a lot of iterations anyway, and hence the advantage of WSS 32 is marginal.

The next question, which naturally arises from the observations above, is whether the number of iterations used in WSS 7 can be reduced by combining WSS 7 with some methods that mimic WSS 32 on the inner support vectors. Here, Figure 11 shows that the number of iterations can be reduced by such combinations in a few cases, but this never pays off in terms of run time, if one considers the entire grid. On the grid points with small validation error, however, the situation is slightly more involved. Clearly, the combination with WSS 2048 performs worst, yet combining WSS 7 with WSS 512 or WSS 1024 sometimes yield a shorter run time. Finally, Figure 16 shows

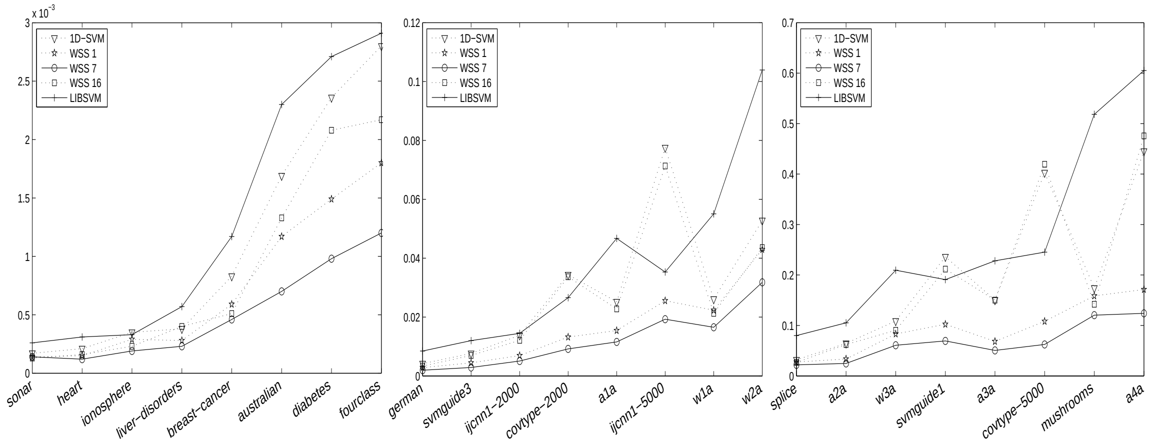

Figure 6: Average run time in seconds per grid point of LIBSVM and MVP compared to some other approaches over the entire 10 by 10 grid. The 2D-MVP approach of WSS 16 is not a good alternative to the 1D-SVM or even the two-dimensional WSS 7. Moreover, LIBSVM is significantly slower than WSS 7.

that, at least for the data set SVMGUIDE1, the improvements achieved by these combinations are mainly at grid points that do not require a lot of iterations. On the other hand, it also illustrates that the computational overhead of these combinations is significant.

Finally, let us compare LIBSVM with some subset selection strategies such as the MVP approach of WSS 16 and the overall best performing WSS 7. Here, see Figure 6 for a short impression and Figures 17 to 19 for the details, the most interesting observation is that although WSS 1 and LIBSVM have comparable behavior in terms of iterations, they substantially differ in terms of their run time. Because we used our own implementation of LIBSVM's solver, which employed the same SSE2 optimizations as the 2D-SVM methods, the only way to explain this behavior is that the subset selection strategy of LIBSVM is significantly more expensive than the simple WSS 1. To understand the latter, recall that LIBSVM's strategy is based on computing an approximate 2D-gain, which is quite expensive as we have seen in Figures 11 and 12 for the 2D-SVM methods WSS 32, WSS 64, WSS 128, and WSS 256. In addition, LIBSVM's strategy cannot be efficiently vectorized, which is another disadvantage compared to WSS 1. Finally, it is interesting to note that WSS 7 is between 2 and 4 times faster than LIBSVM, when the average over all grid points is considered. Moreover, on the grid points with small cross validation error, the improvement is rarely less than a factor of 4, and as Figure 19 illustrates, this is most likely not an artifact caused by different optimal grid points. Indeed, on some grid points LIBSVM needs more than 10 times the run time WSS 7 requires.

5.4 Influence of the Stopping Criterion

In this subsection, we investigate the influence of the stopping criterion (9) on the computational requirements. To this end, we considered the 10-fold cross validation procedure described earlier. Moreover, in order to save time, we only considered the best performing working set selection strategy, namely WSS 7. For this method, we considered our stopping criterion (9) and the classical duality gap stopping criterion (7), where we set the right hand side of both stopping criteria to be $\varepsilon/(2\lambda)$ with $\varepsilon := 0.001$. Note that this is exactly the same set-up as in our previous experiments, and it is not hard to show that for the duality gap (7), this choice again leads to the same theoretical bounds on the generalization performance.

The results of our experiments are summarized in Figures 20 to 22. A quick look shows that, not surprisingly, the stopping criterion (9) never leads to more iterations, but the improvements depend very much on the data set. Moreover, these smaller number of iterations also pay off in terms of run time, though the effect is less pronounced when we consider the entire grid. We believe this is due to the fact that computing (9) is a little more expensive than computing (2), since it involves two instead of just one clipping operations. In this regard, it is interesting to note that the SSE2 instruction set in emmintrin.h makes it possible to avoid expensive branches for the computation of the clipping by providing $min()$ and $max()$ operations. In turn, this results in a relatively cheap stopping criterion; from some ad-hoc measurements made for a different purpose, we estimate that this computation costs about 10% of an entire iteration, though the exact numbers are most likely hardware dependent. When we only consider the grid points with small validation error, the positive effect of the clipped duality gap is amplified as Figure 21 shows. The reason for this behavior is illustrated in Figure 22 for the SVMGUIDE1 data set. Indeed, this figure shows that for small values of λ, the stopping criterion (9) leads to both substantially less iterations and shorter run times, whereas for larger λ, the computational requirements for both stopping criteria are essentially the same. Although uniformly superior, the positive effect of using (9) is thus highly inhomogeneously distributed over the parameter grid.

5.5 Comparing Different Numbers of Nearest Neighbors

So far we have considered WSS 7 for 10 nearest neighbors only. Of course, this was a relatively arbitrary choice, and hence it is interesting to investigate how the computational requirements change with the number of nearest neighbors. This is the goal of this subsection.

To this end, we again used the 10-fold cross validation procedure described earlier. Moreover, we considered the behavior of WSS 7 for N-nearest neighbors, where $N = 5, 10, 15, 20, 25, 30$. Our first observation was that, for $N = 25$ and $N = 30$, there was rarely an improvement in terms of iterations, but the required run time tended to slightly increase compared to smaller N. To keep the figures clean, we hence plotted the results for $N = 5, 10, 15, 20$, only. Figures 23 and 24 show that WSS 7 behaves slightly worse for $N = 5$, while for larger N the behavior over the *entire* grid is essentially indistinguishable. The latter observation mildly changes, if one only considers the hyper-parameters with small cross validation error, yet it is unclear to which extend this effect is caused by possibly different hyper-parameters picked by the different methods. In addition, a detailed look at Figure 25 does not really clarify the situation, since many of the run times measured are close to the finest resolution of time.h. Consequently, it seems safe to say that, at least in the range $N = 10 \dots 20$, the performance of WSS 7 is essentially independent of N.

5.6 Comparing Different Initializations

Let us now investigate the influence of different initialization strategies on the computational requirements. To this end, we trained 2D-SVM with WSS 7 and with different combinations of cold and warm start options on the data sets summarized in Table 1. Moreover, we again used the 10-fold cross validation procedure described earlier.

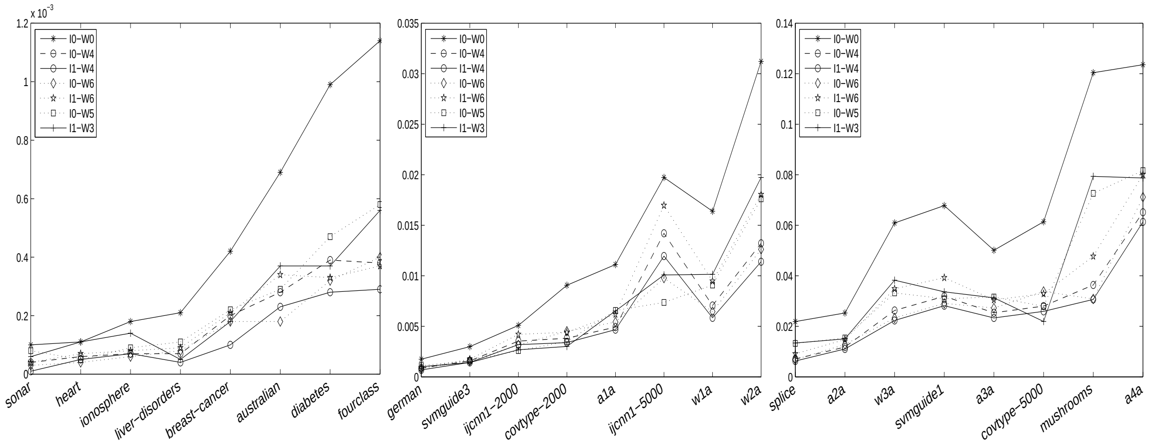

Figure 7: Average computational requirements per grid point of more complex initialization strategies for the 2D-SVM with WSS 7 for small (left), mid-sized (middle), and relatively large data sets (right). The graphics display the run time in seconds. The cold start initializations with zeros (I0- plots) need less iterations but in most cases more run time. In almost all cases, the more complicated initialization strategies perform better than the simple warm start approaches. Overall, I0-W4, I1-W4, and I0-W6 are the most efficient methods in terms of run time.

The first observation is, see Figure 26 for details, that initializing with zeros always leads to less iterations than initializing with a kernel rule. Surprisingly, however, the required run time for both initialization strategies is substantially less different. A closer inspection revealed[10] that this is caused by the fact that the solver initialized with the kernel rule method I1-W1 spends most of its iterations during initialization, that is, most of the iterations counted are from the outer loop of Procedure 4. Since these iterations do not involve the working set selection and the computation of the stopping criterion, they are relatively cheap compared to the iterations of the actual solver described in Algorithm 2. Moreover, Figure 26 shows that the simple warm start strategies W2, W3, and W5 do reduce the computational requirements significantly, where in almost all cases the scaling approach of W3 and W5 performs superior.

Interestingly, the computational requirements can often be further reduced by one of the more complicated initialization strategies W4 and W6 as Figure 7 illustrates, see also Figure 27 for more details. In particular, the combinations I0-W4, I1-W4, and I0-W6 run in most cases faster than the simple combination I0-W3, and overall it seems fair to say that I1-W4 performs best. However note that this approach requires access to the entire kernel matrix, and hence the combinations I0-W4 and I0-W6 may be the better choice, if storing this matrix is not an option.

We also conducted a control experiment in which the warm start options available for SVMs with offset are compared. Figure 28, which displays the corresponding results, shows that in most cases scaling by W3 and W5 is better than keeping the solution, that is, W2. This is similar to our results for SVMs without offset, but a closer look reveals, that the run time gain for SVMs with offset is both less pronounced and less consistent. In particular for the larger data sets, the gain by

[10] For brevity's sake we omitted the display of the corresponding plots.

using a warm start for SVMs with offset is about 20%, while for SVMs without offset it is about 45% even if only the simple warm start option W5 is used. Moreover, the more complex strategies for SVMs without offset can reduce the run time by about 60% on these data sets. Consequently, it seems fair to say that SVMs without offset benefit substantially more from warm start strategies than SVMs with offset do.

Let us finally investigate the effect of some of the initialization strategies for different hyper-parameter pairs. Here Figure 29 reveals that the warm start options perform almost uniformly better than the cold start option I0-W0. Moreover, the complex warm start strategy W4 achieves a significant gain for small values of λ. Since these λ are computationally more demanding than large values of λ, the success of W4 can be explained. On the other hand, the strategies W5 and W6 start with the smallest value of λ, and hence they do not achieve any improvement over I0-W0 for this λ. However, they achieve a significant improvement for basically all other values of λ, which in turn explains their success. By combining these observations and the fact that the cold start I0 requires a relatively small number of iterations on medium values for λ, it thus seems promising to use a hybrid strategy that starts with such a medium value for λ, and then performs W4 for smaller λ and W6 for larger values. However, investigating such a strategy is out of the scope of this paper.

6. Conclusions

We have thoroughly investigated SVMs without offset term *b* that use the hinge loss and Gaussian kernels. It turned out that these SVMs have convergence rates and classification performance that are comparable to SVMs with offset, while the absence of the offset gives more freedom in the algorithm design. In particular, we identified three areas, where this additional freedom results in faster algorithms:

- **Working set selection.** In principle, an SMO-type solver for SVMs without offset can update one variable at each iteration. However, we saw that this approach does not lead to run times that were shorter than those of an SMO-type solver for SVMs with offset. We then identified some selection methods for working sets of size two, that modified the search for working sets of size one only very slightly. It further turned out that these modifications decreased the number of iterations substantially, and since updating the gradient and computing the stopping criterion for two variables did not change the costs of an iteration dramatically, these modification also resulted in a significantly shorter run time. It is further worth mentioning that the most successful selection strategies for workings sets of size two were actually combinations of a couple of such simple modifications. The reason for the latter was that some strategies worked particularly well for large values of the regularization parameter λ, while others worked better for small values of λ. The good combinations then contained both types of strategies and identified the better one at each iteration automatically.

- **Stopping criterion.** Another improvement of the run time behavior of our algorithm came from a new stopping criterion that has a clear justification from recent statistical analysis of SVMs. This stopping criterion, which is essentially a relaxed duality gap, never leads to more iterations than the classical duality gap stopping criterion, but it often decreased the number of iterations. Moreover, its computational costs were almost identical to those of the classical duality gap, and hence it often resulted in shorter run times.

- **Warm start initializations.** SVMs without offset also allow more freedom in the design of warm start initializations when the hyper-parameters are determined over a grid of hyper-parameters. We investigated a couple of such initialization methods and saw that some of them led to a substantially shorter run time. Moreover, by comparing to some warm start initializations for SVMs with offset, we observed that SVMs without offset benefit significantly more from such strategies.

It seems fair to remind the reader that in our experiments we only considered data sets for which the kernel matrix fit completely in the RAM of a desktop computer. With present configurations of, say up to 8GB RAM, this limits the data set size somewhere between 25,000 and 30,000 samples. While such sizes are typically not considered to be extremely large, they already constitute a decent challenge for existing off-the-shelf SVM software, if the training time is an issue. Moreover, even for smaller data sets a fully automated hyper-parameter selection run for SVMs with offset is, for some applications, too time intensive. Our new SVM solver yields a significant reduction in time for medium-sized data sets, thus opening the applicability of SVMs to such problem domains. However, it seems fair to say that although many data sets actually fall in this range of size, some other applications demand processing substantially larger data sets. So far, it remains unclear, how well our new solver performs for such data sets, and since our experimental study was already quite extensive and expensive, we postpone this question to future research.

Some other directions of future research include the following questions: *a)* Are there cheap modifications of our 2D-working set selection strategy that identify working sets of larger size for which the number of iterations and the run time is further reduced? *b)* Can some of our ideas be used or modified for other SVMs, that, for example, use different kernels and/or loss functions? *c)* Can the run time of the solver be further improved by not only using vectorization via SSE instructions but by also distributing tasks between different cores of a modern processor?

Appendix A. Detailed Experimental Results

On the following pages, we present more graphics illustrating our experimental findings. To keep these graphics in order, we divided the appendix in several subsections, which follow the order of the subsections of Sections 5.

A.1 Results for the Different Working Set Selection Methods

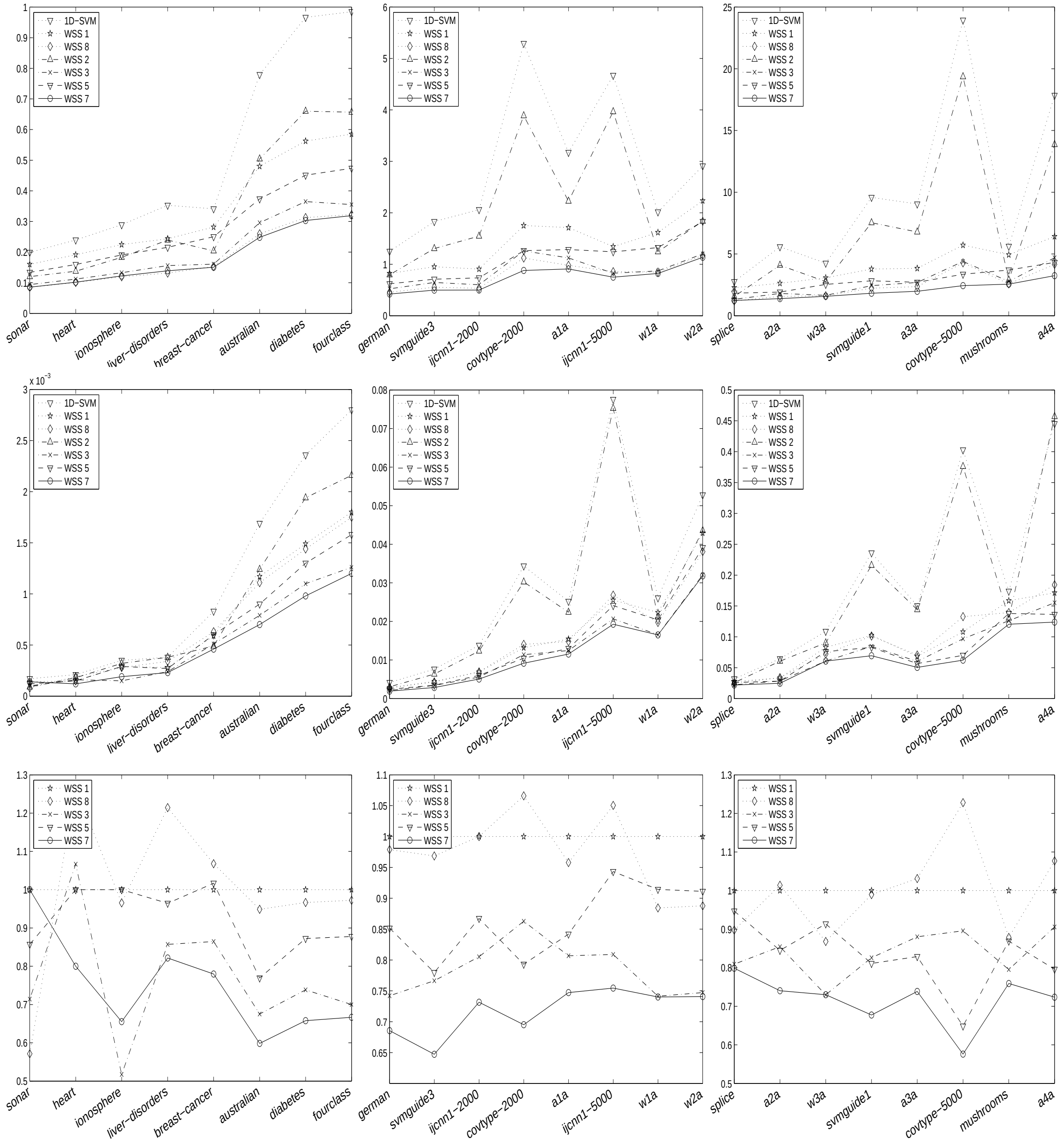

Figure 8: Average computational requirements per grid point of simple extensions of the 1D-search strategy over the entire 10 by 10 grid. The graphics display the number of iterations in thousands (top), the run time in seconds (middle), and the ratios WSS x/WSS 1 of the run times (bottom). WSS 7 performs almost uniformly best in both metrics.

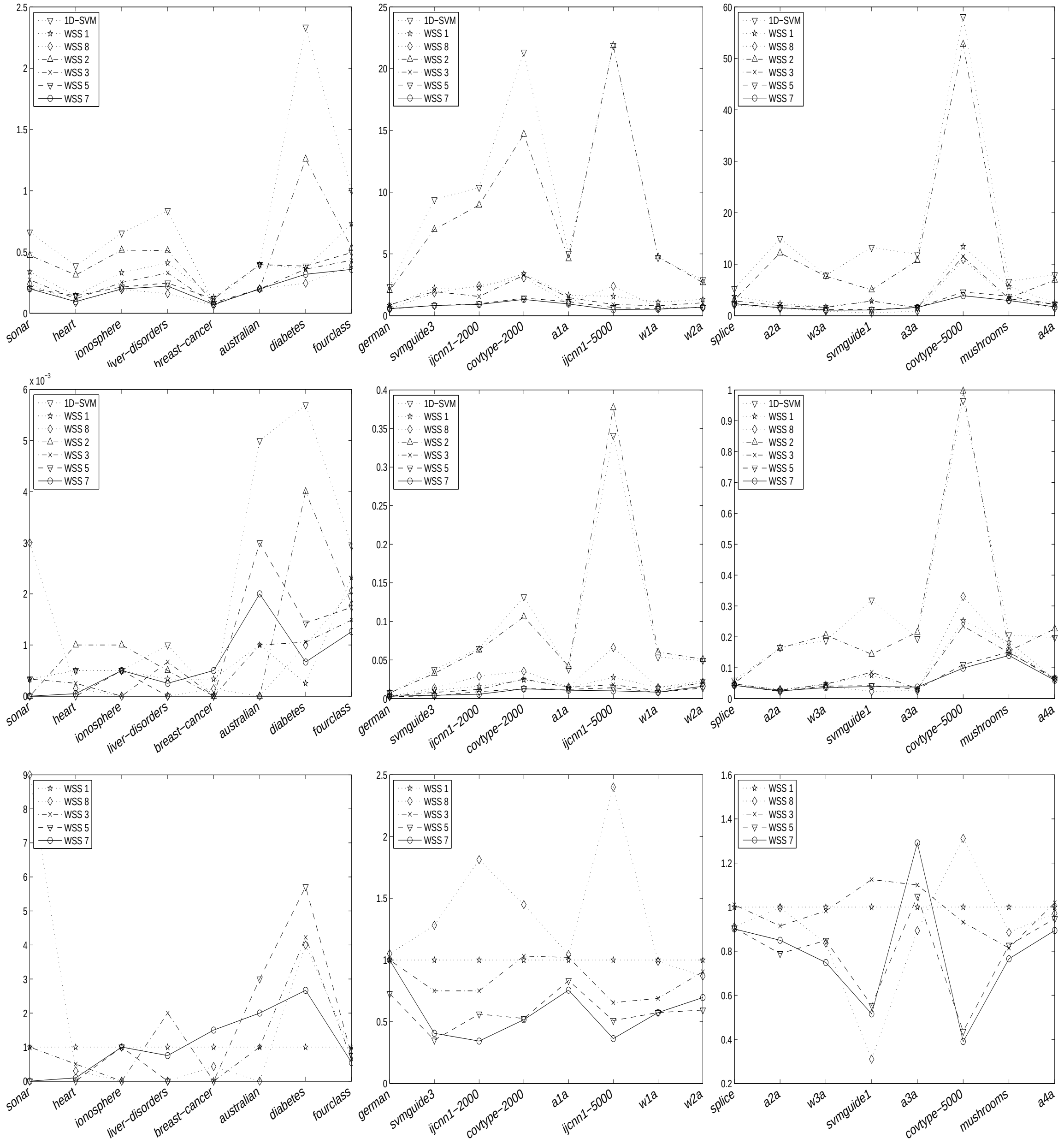

Figure 9: Computational requirements of simple extensions of the 1D-search strategy on the grid points whose cross validation error is not larger than 1.05 the minimal cross validation error. The graphics display the average number of iterations in thousands (top), the run time in seconds (middle), and the ratios WSS x/WSS 1 of the run times (bottom). Unfortunately, for the small data sets, the run time measurements are not very reliable. In addition, the set of considered grid points may slightly vary for the different methods, which in turn may influence the computational requirements and hence the graphic at the bottom left has little informative value. It seems fair to say that overall, WSS 7 performs best in both metrics, but is closely followed by WSS 5 in terms of run time.

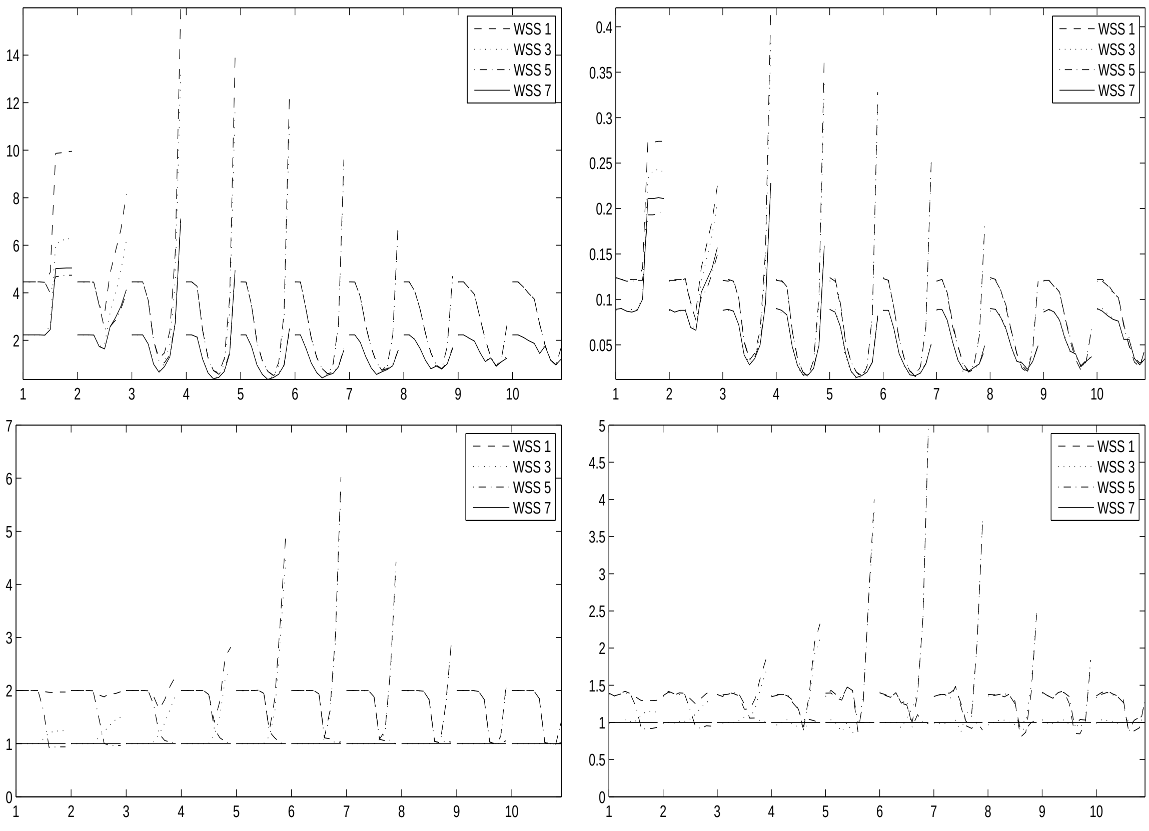

Figure 10: Computational requirements per single grid point of simple extensions of the 1D-search strategy for the SVMGUIDE1 data set. Each horizontal cell numbered by 1 to 10 corresponds to a single kernel parameter σ and an ordered run through the 10 λ-values, where the left of each cell corresponds to the largest λ-value, and the right to the smallest. Analogously, cell 1 corresponds the the largest σ-value, and cell 10 on the right corresponds to the smallest σ-value. The graphics at the top display the number of iterations in thousands (left) and the run time in seconds (right), both averaged over the 10 folds, for WSS 1, WSS 3, WSS 5, WSS 7, and WSS 8. WSS 7 performs almost uniformly the best in both metrics. However, for large λ, WSS 3 performs comparable, while for small λ, WSS 7 is closely followed by WSS 5. The graphics at the bottom show the ratios WSS x/WSS 7, x$= 1, 3, 5, 7$, for the number of iterations (left) and the run time (right) to illustrate the performance gain of WSS 7.

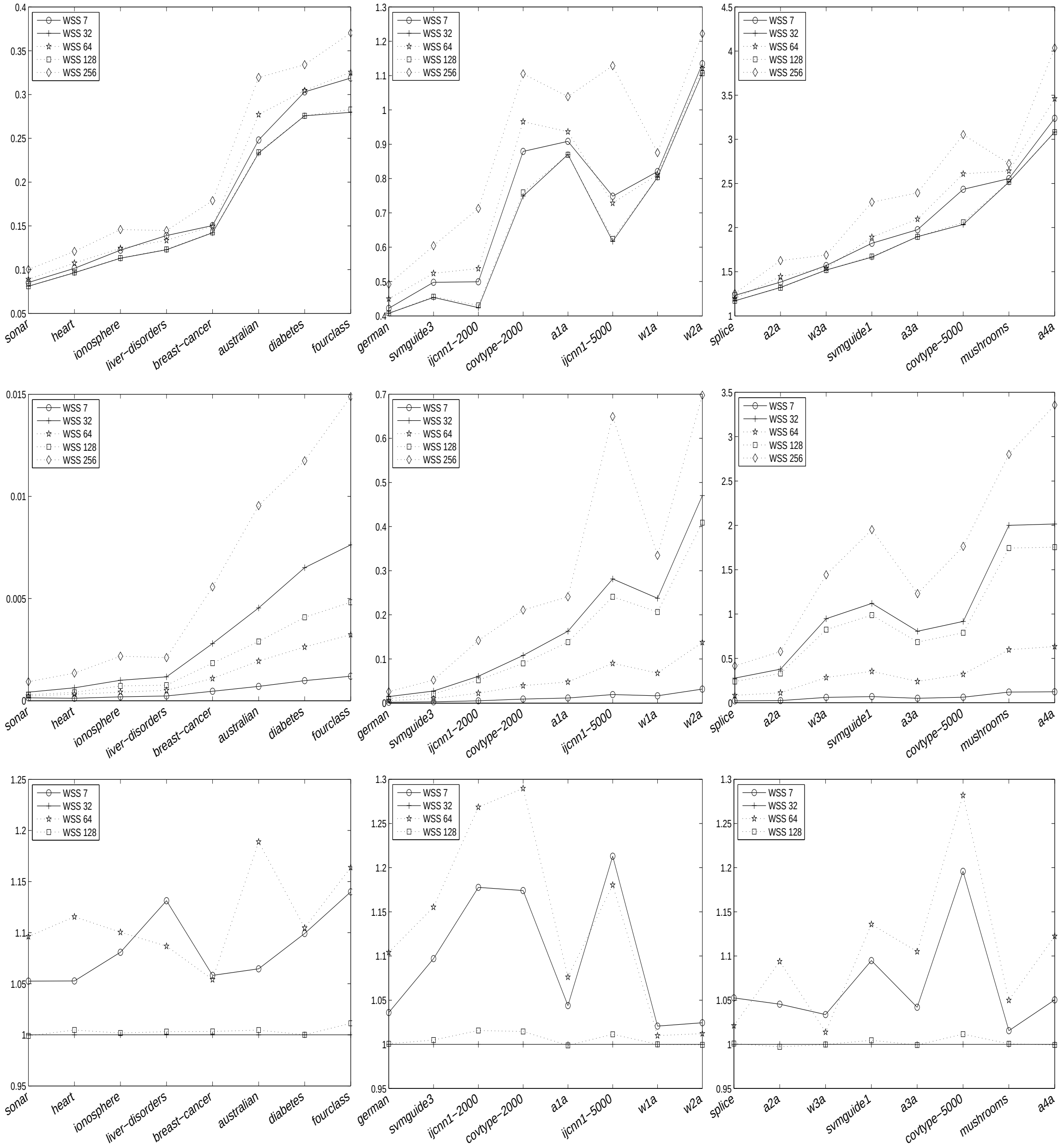

Figure 11: Average computational requirements per grid point of methods based on approximations of the maximal gain strategy WSS 0. The graphics display the number of iterations in thousands (top), the run time in seconds (middle), and the ratios WSS x/WSS 32 of the number of iterations (bottom). Although WSS 7 and the semi-random WSS 64 need slightly more iterations than WSS 32 and WSS 128, their costs per iteration is substantially less, which results in a significantly shorter run time. The completely random WSS 256 needs over-proportionally more run time, possibly because of the less effective hardware cache.

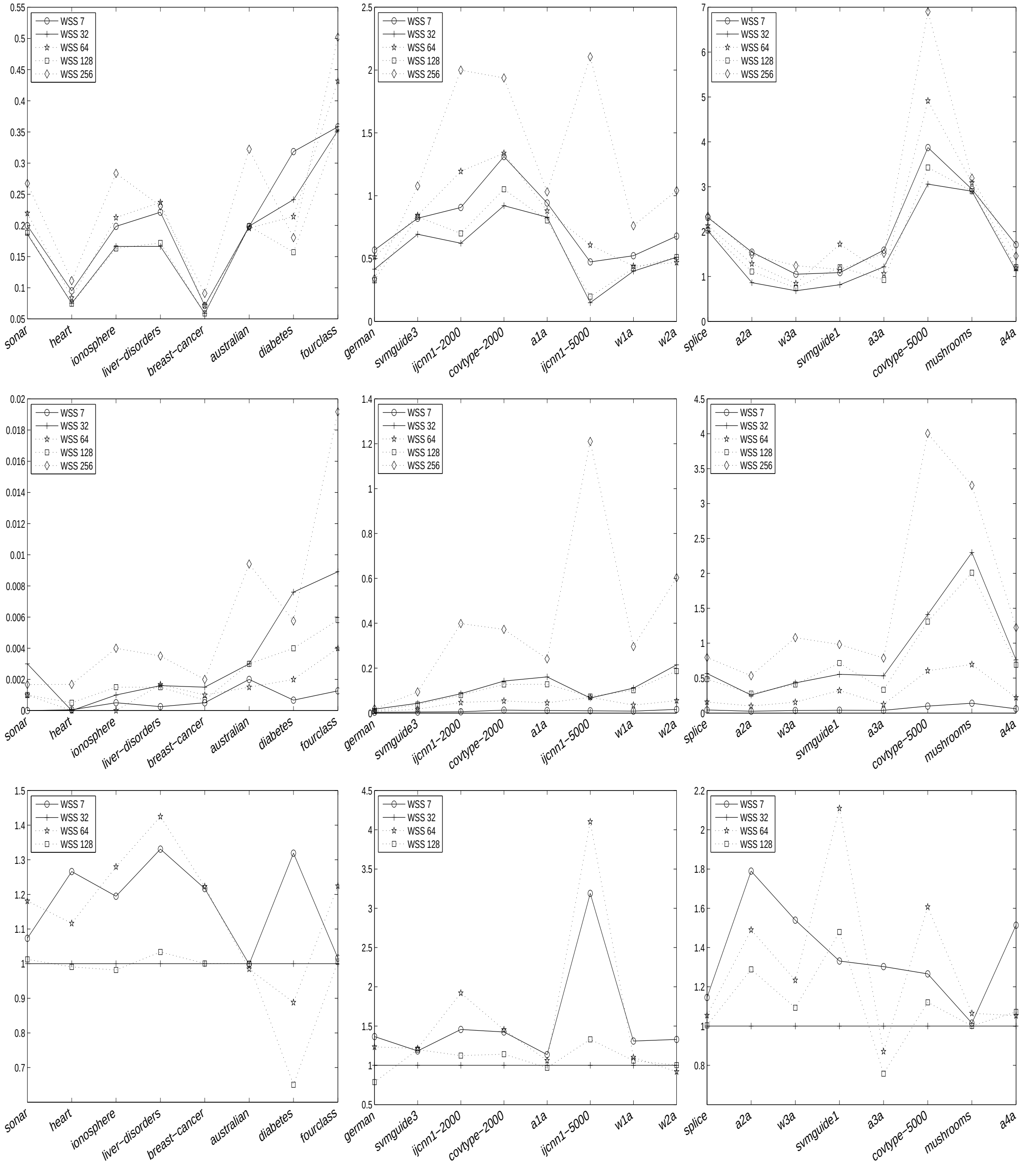

Figure 12: Computational requirements of methods based on approximations of the maximal gain strategy WSS 0 on the grid points whose cross validation error is not larger than 1.05 the minimal cross validation error. The graphics display the average number of iterations in thousands (top), the run time in seconds (middle), and the ratios WSS x/WSS 32 of the number of iterations (bottom). For the small data sets, the run time measurements are not very reliable. In addition, the set of considered grid points may slightly vary for the different methods, which in turn may influence the computational requirements.

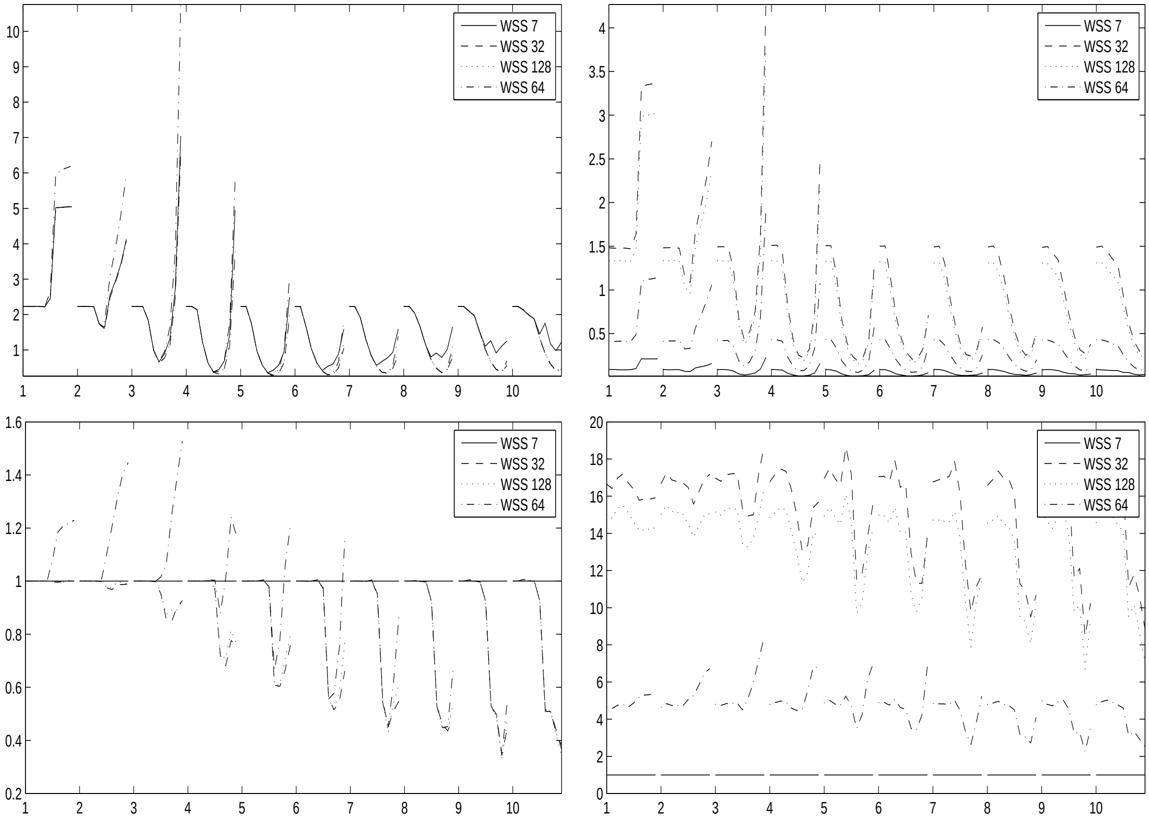

Figure 13: Computational requirements per single grid point of methods based on approximations of the maximal gain strategy WSS 0 for the SVMGUIDE1 data set. The four graphics have the same format as the ones in Figure 10. The graphics at the top display the number of iterations in thousands (left) and the run time in seconds (right), both averaged over the 10 folds, while the graphics at the bottom display the corresponding ratios WSS x/WSS 7. For some grid points, WSS 7 and WSS 32 need approximately the same number of iterations, while for some other grid points, WSS 7 needs significantly more. Nonetheless, the run times of WSS 32 are substantially worse than that of WSS 7.

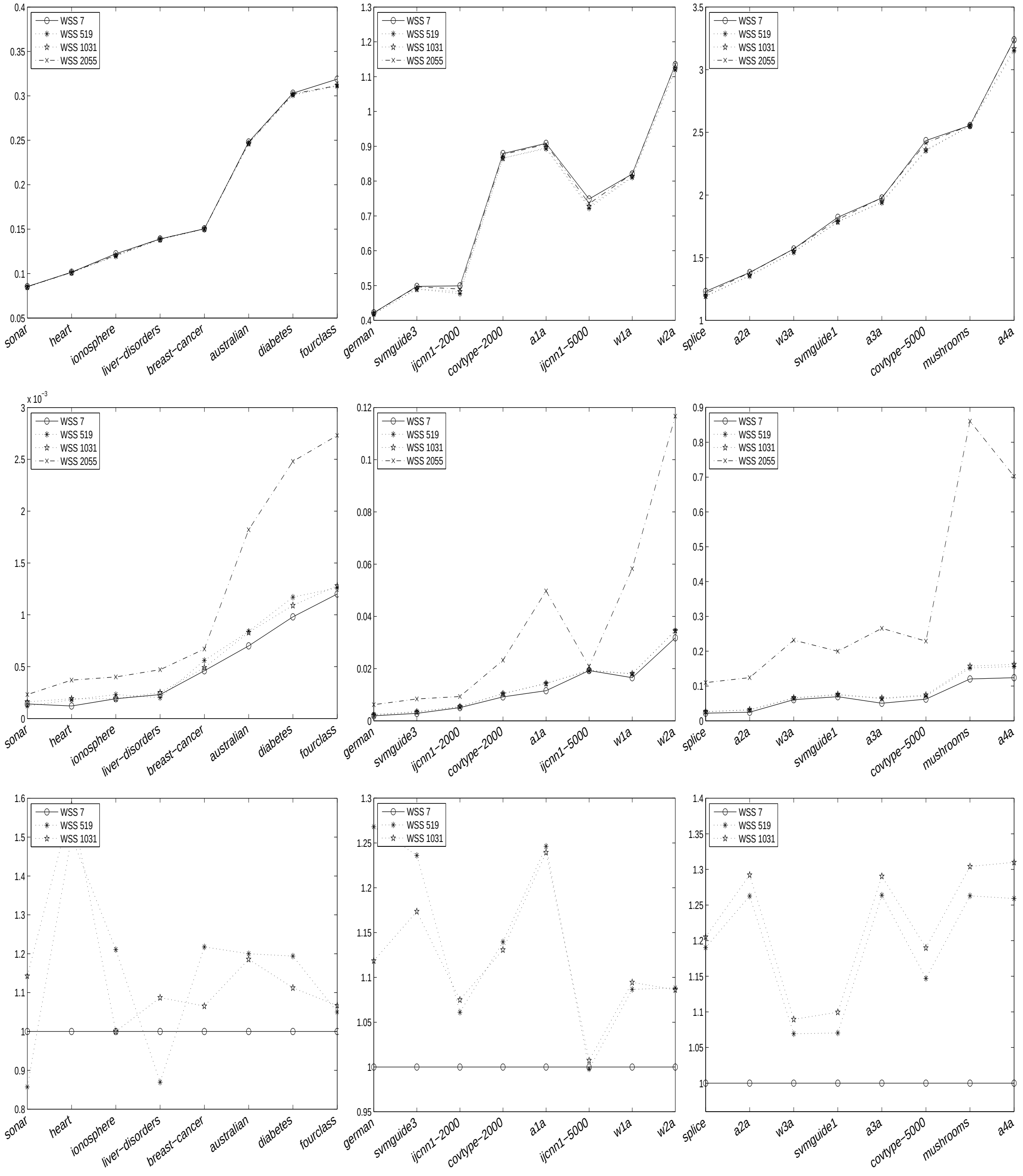

Figure 14: Average computational requirements per grid point of combining WSS 7 with some methods that use the formula for the approximate gain on inner SVs over the entire 10 by 10 grid. The graphics display the number of iterations in thousands (top), the run time in seconds (middle), and the ratios WSS x/WSS 7 of the run times (bottom). Although the combinations need a slightly smaller number of iterations, their additional overhead per iteration leads to longer run times.

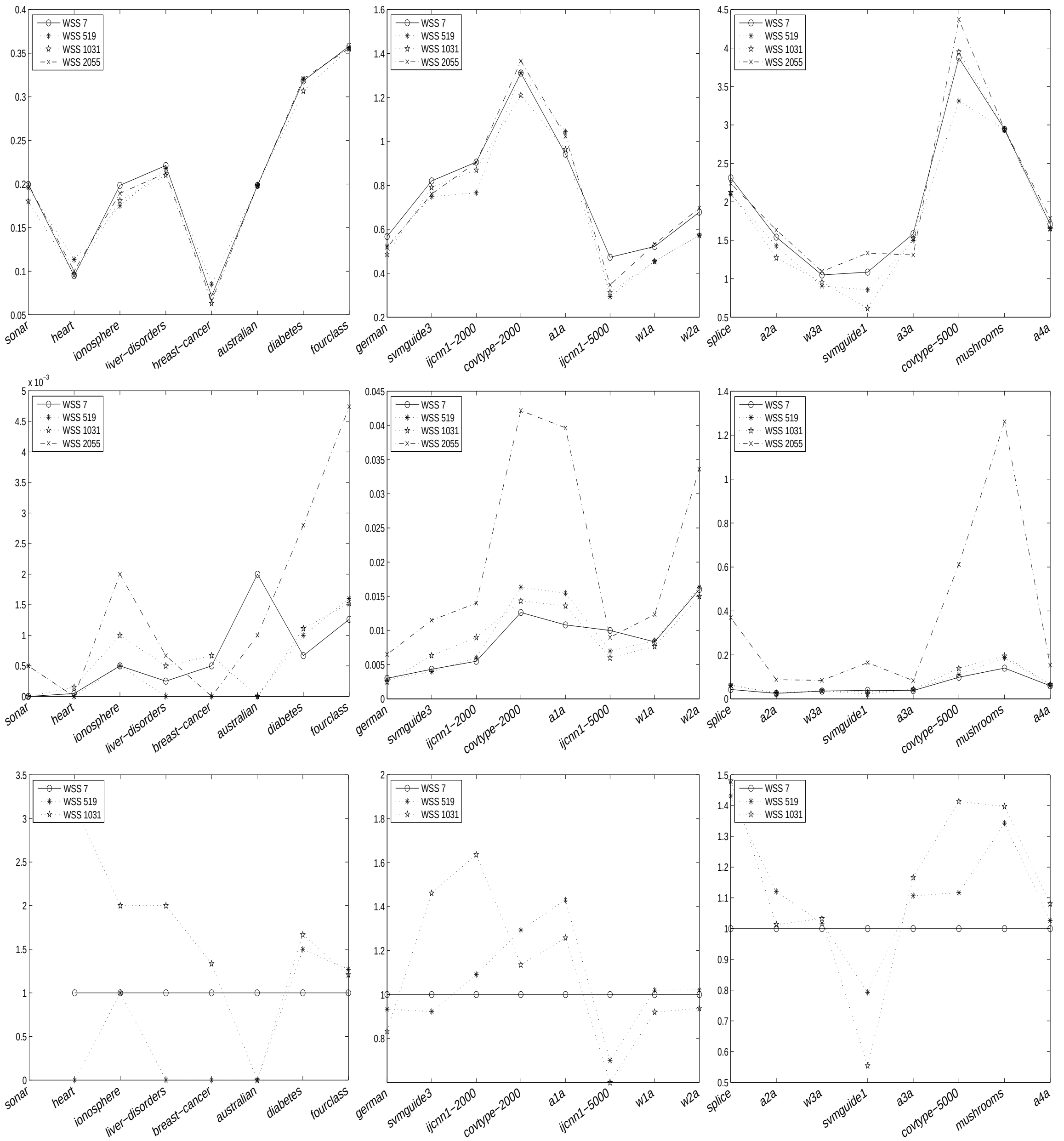

Figure 15: Computational requirements of combining WSS 7 with some methods that use the formula for the approximate gain on inner SVs on the grid points whose cross validation error is not larger than 1.05 the minimal cross validation error. The graphics display the number of iterations in thousands (top), the run time in seconds (middle), and the ratios WSS x/WSS 7 of the run times (bottom). For the small data sets, the run time measurements are not very reliable. In addition, the set of considered grid points may vary slightly for the different methods, which in turn may influence the computational requirements.

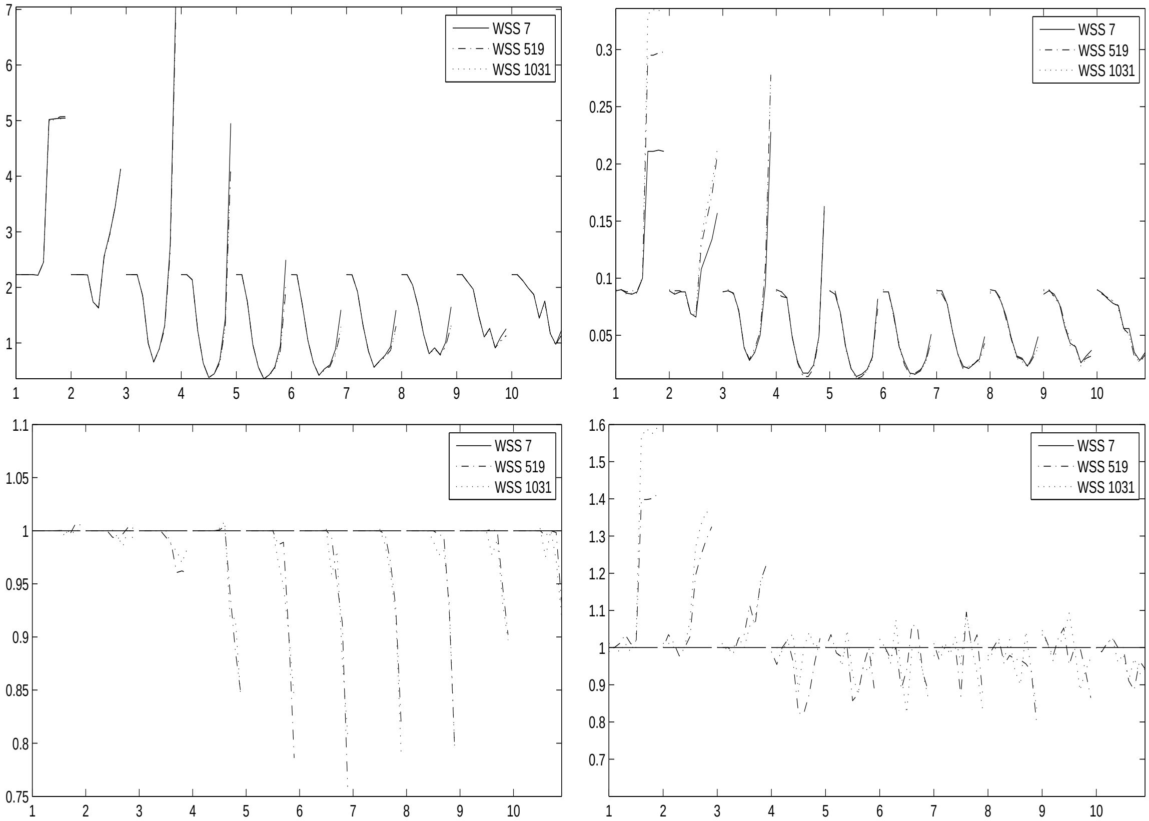

Figure 16: Computational requirements per single grid point of methods based on simple extensions of the 1D-search strategy for the SVMGUIDE1 data set. The four graphics have the same format as the ones in Figure 10. The graphics at the top display the number of iterations in thousands (left) and the run time in seconds (right), both averaged over the 10 folds, while the graphics at the bottom display the corresponding ratios WSS x/WSS 7. Note that for large λ the Boolean flag of WSS 4 is typically not set to true during the optimization, and hence all methods reduce to WSS 3. Analogously, for large λ and σ, the graphics nicely display the additional costs of WSS 512 and WSS 1024. Finally, the differences in the run time occur on a very low and hard to measure level, which explains the fluctuations in the bottom right graphics.

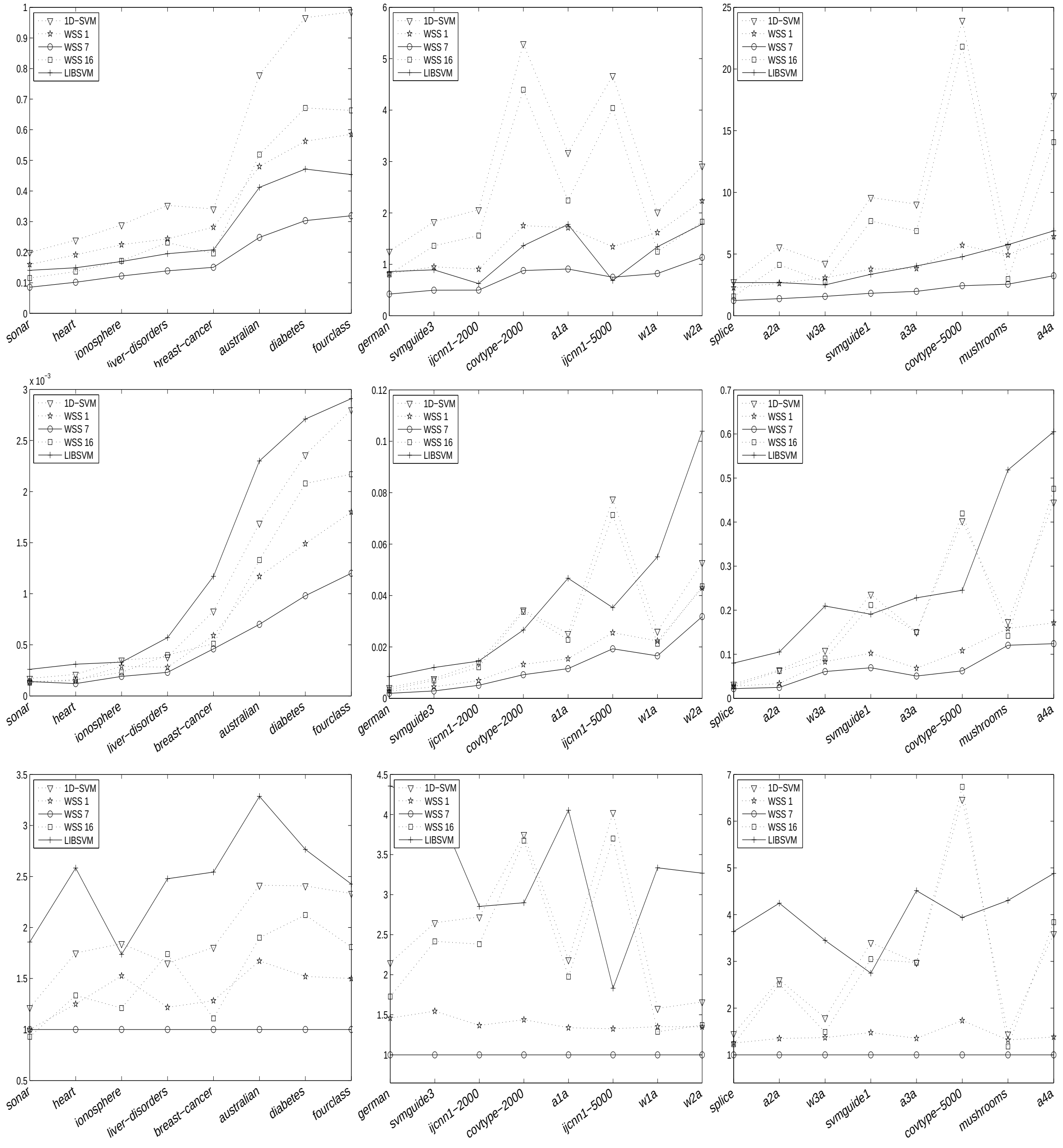

Figure 17: Average computational requirements per grid point of LIBSVM and MVP compared to some other approaches over the entire 10 by 10 grid. The graphics display the number of iterations in thousands (top), the run time in seconds (middle), and the ratios x/WSS 7 of the run times (bottom). The 2D-MVP approach of WSS 16 is not a good alternative to the 1D-SVM or even the two-dimensional WSS 7. Moreover, although WSS 1 and LIBSVM perform approximately the same number of iterations, their run time is significantly different due to the more expensive working set strategy of LIBSVM.

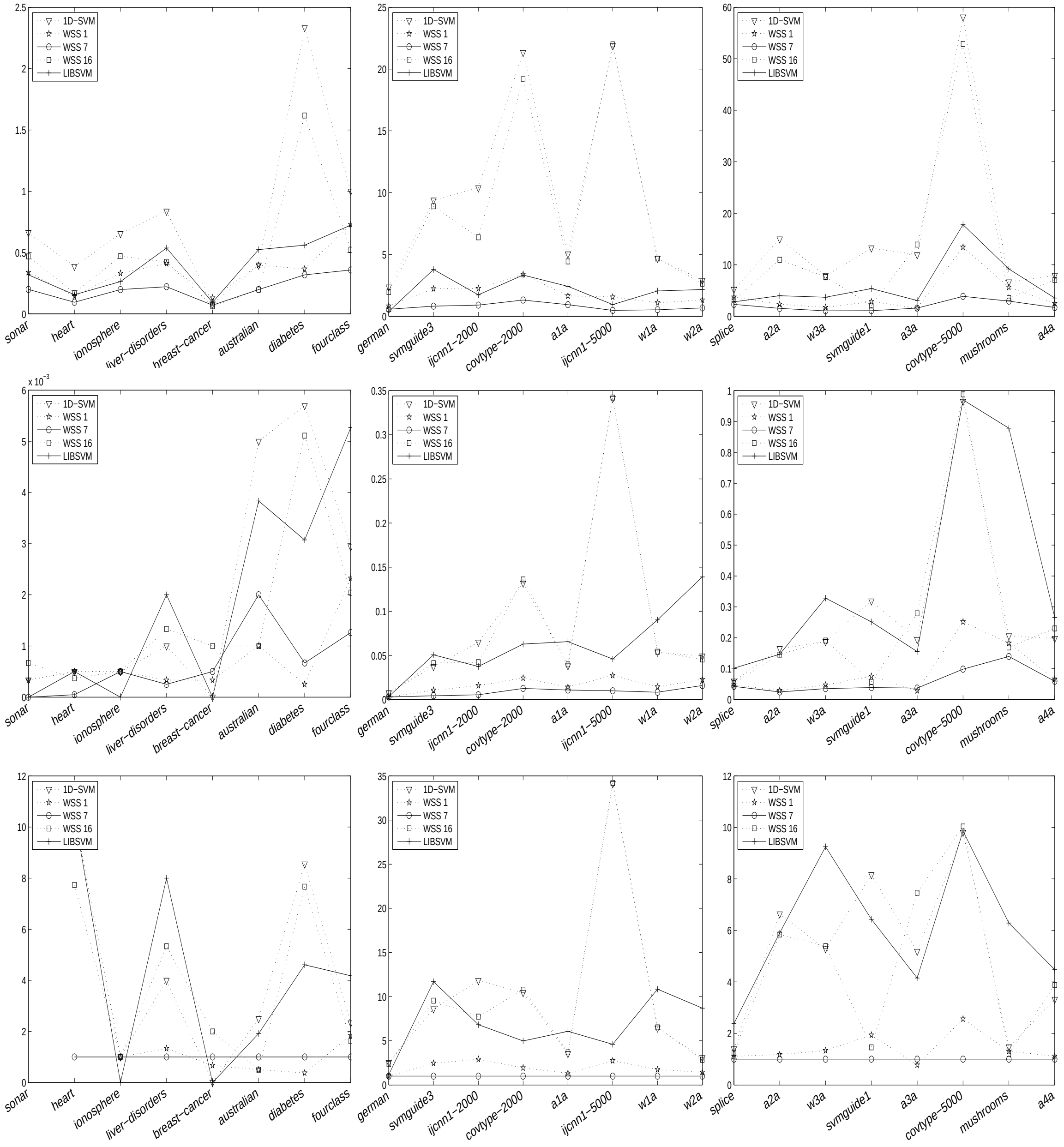

Figure 18: Computational requirements of LIBSVM and MVP compared to some other approaches on the grid points whose cross validation error is not larger than 1.05 the minimal cross validation error. The graphics display the average number of iterations in thousands (top), the run time in seconds (middle), and the ratios x/WSS 7 of the run times (bottom). Again, for the small data sets, the run time measurements are not very reliable. In particular, for the SONAR data set, the average *measured* run time for WSS 7 was 0.00 seconds, and hence the corresponding ratios could not be plotted. Besides that the conclusions of Figure 17 are confirmed.

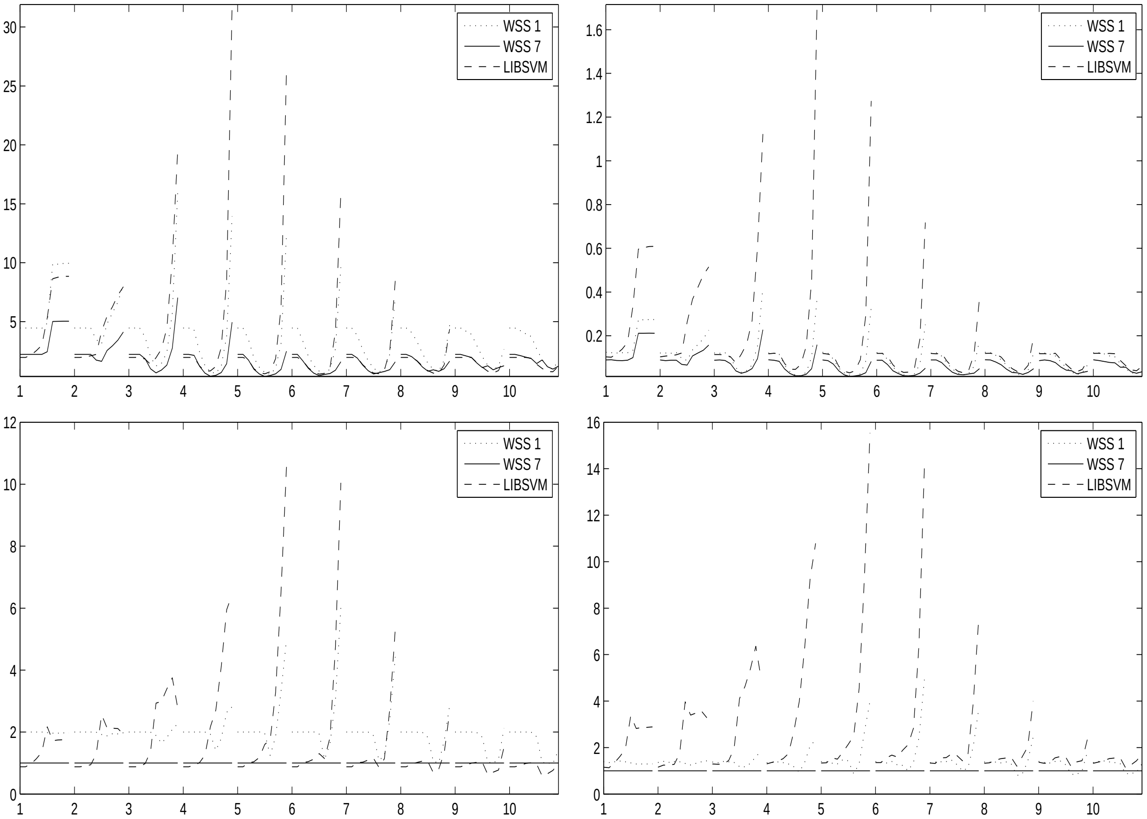

Figure 19: Computational requirements per single grid point of methods based on simple extensions of the 1D-search strategy and LIBSVM on the SVMGUIDE1 data set. The four graphics have the same format as the ones in Figure 10. For flatter kernels, LIBSVM needs less iterations than WSS 7, possibly because it solves a different optimization problem, however the improvement is small in terms of absolute numbers. On the other hand, both WSS 1 and WSS 7 are less sensitive to small λ values in regions with high computational demand.

A.2 Results for the two Different Stopping Criteria

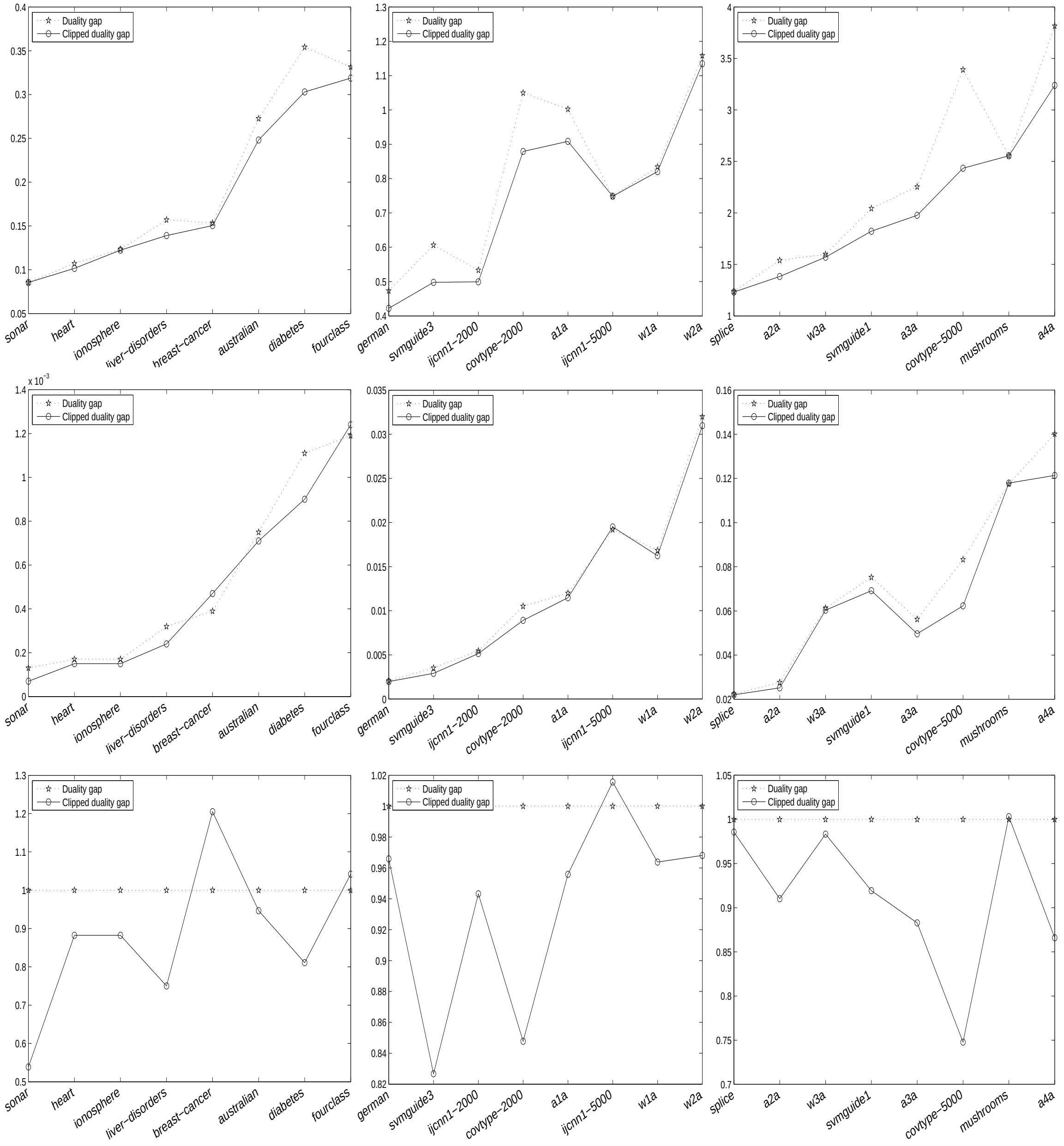

Figure 20: Average computational requirements per grid point of WSS 7 with different stopping criteria. The graphics at the top display the number of iterations in thousands for the 2D-SVM with WSS 7, while the graphics in the middle show the corresponding run time in seconds. The graphics at the bottom display the ratio of run times.

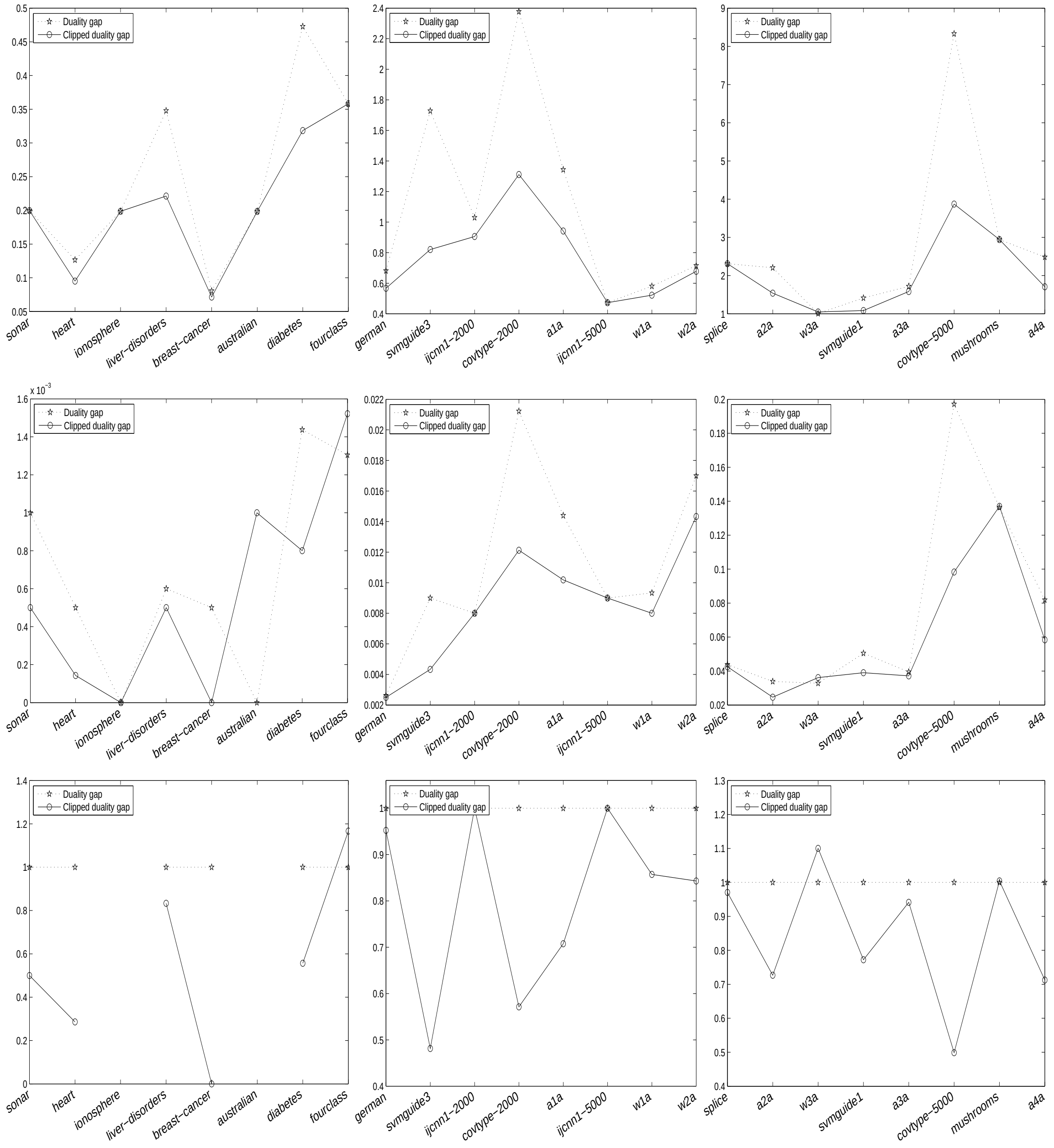

Figure 21: Computational requirements of WSS 7 with different stopping criteria on the grid points whose cross validation error is not larger than 1.05 the minimal cross validation error. Again, the graphics at the top display the number of iterations in thousands for the different stopping criteria applied to the 2D-SVM with WSS 7, while the graphics in the middle show the corresponding run time in seconds. The graphics at the bottom display the ratio of run times, where we note that for some data sets in the bottom left graphic the ratio could not be computed since the *measured* run time was zero.

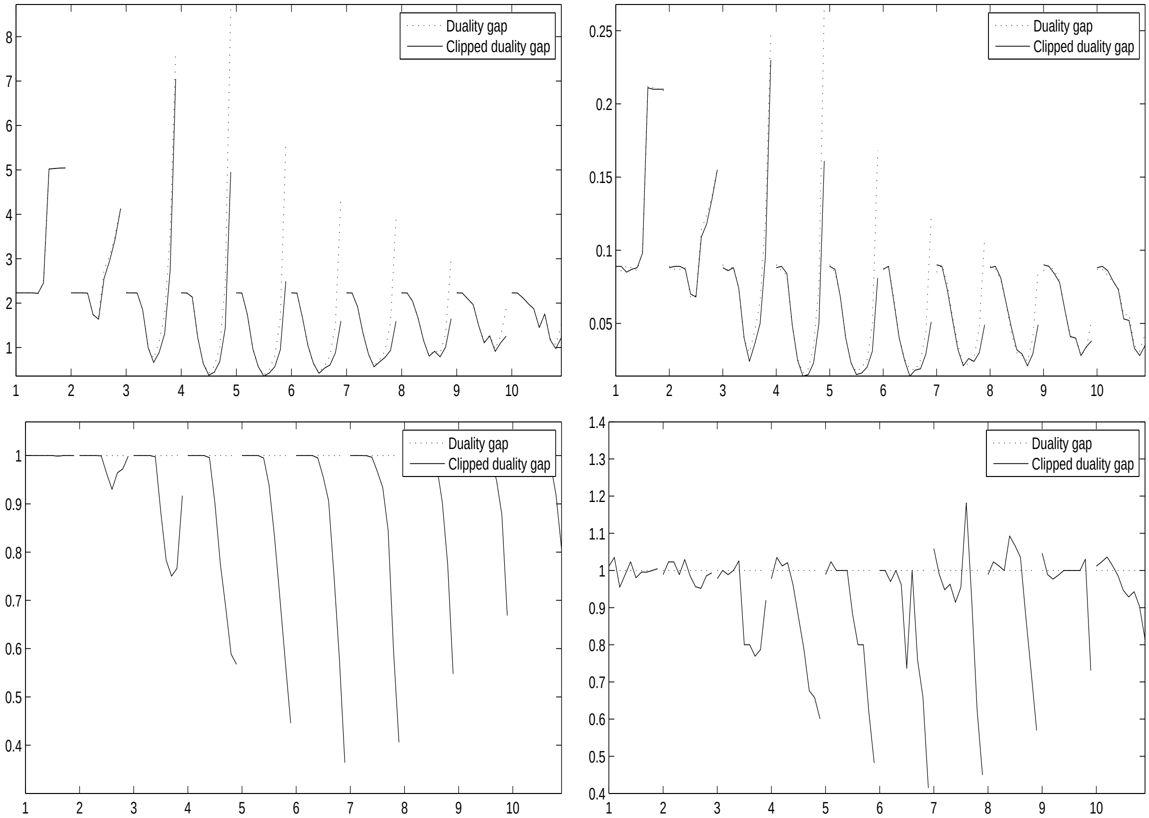

Figure 22: Computational requirements per single grid point of the two stopping criteria for the SVMGUIDE1 data set. The four graphics have the same format as the ones in Figure 10. The graphics at the top display the number of iterations in thousands (left) and the run time in seconds (right), both averaged over the 10 folds, while the graphics at the bottom display the corresponding ratios. The clipped stopping criteria (9) helps for small values of λ, whereas for larger values the behavior is basically identical. Again, some of the roughness in the bottom right graphic can be explained by the resolution of the time measurements. However, the general trend in this graphic is confirmed by the ratio of iterations displayed in the bottom left graphic.

A.3 Results for Different Numbers of Nearest Neighbors

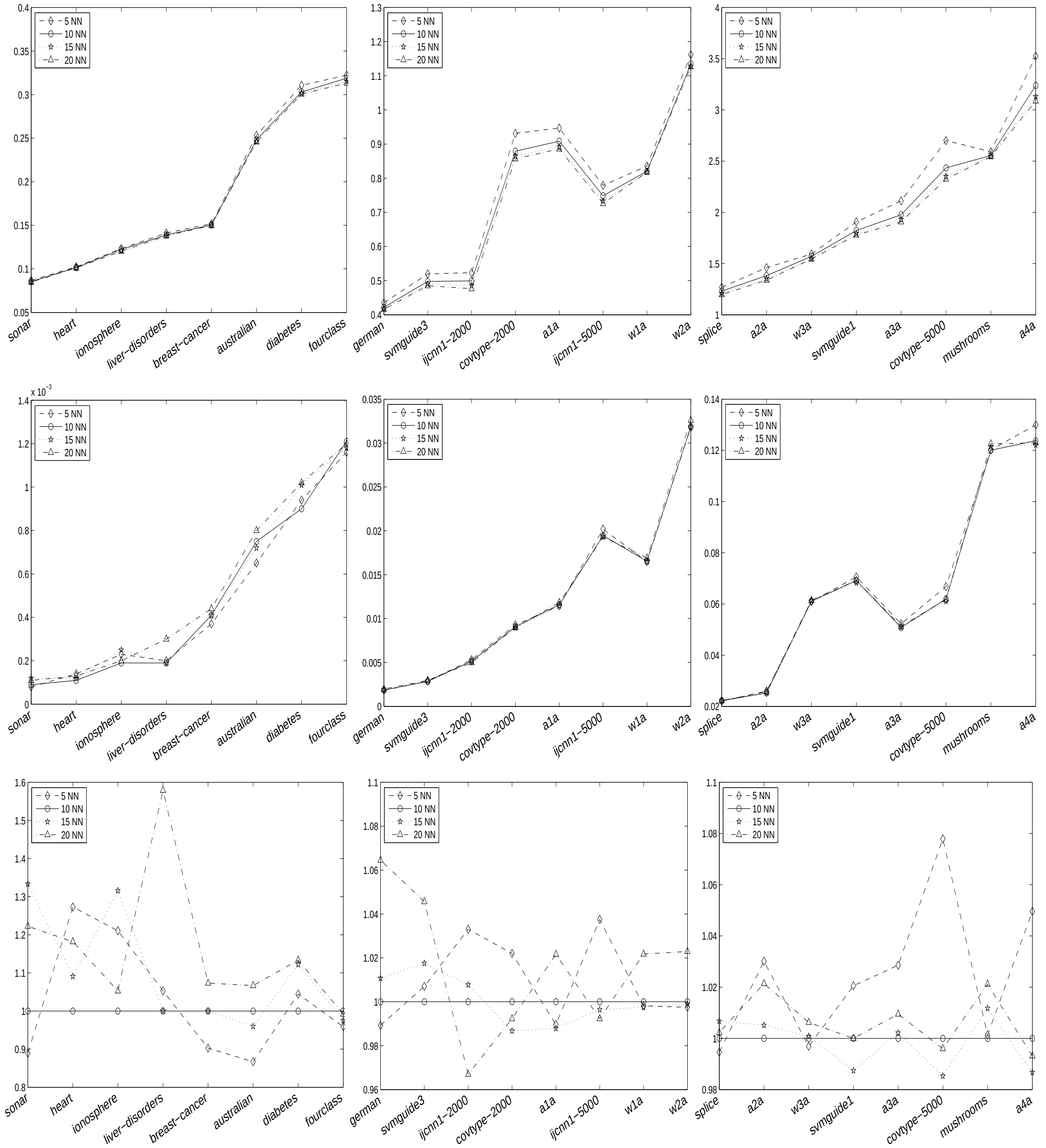

Figure 23: Average computational requirements per grid point of WSS 7 with different numbers N of nearest neighbors. The graphics display the number of iterations in thousands (top), the run time in seconds (middle), and the corresponding ratios $xNN/10NN$ of the run times (bottom). For $N \geq 10$, the performance is basically identical.

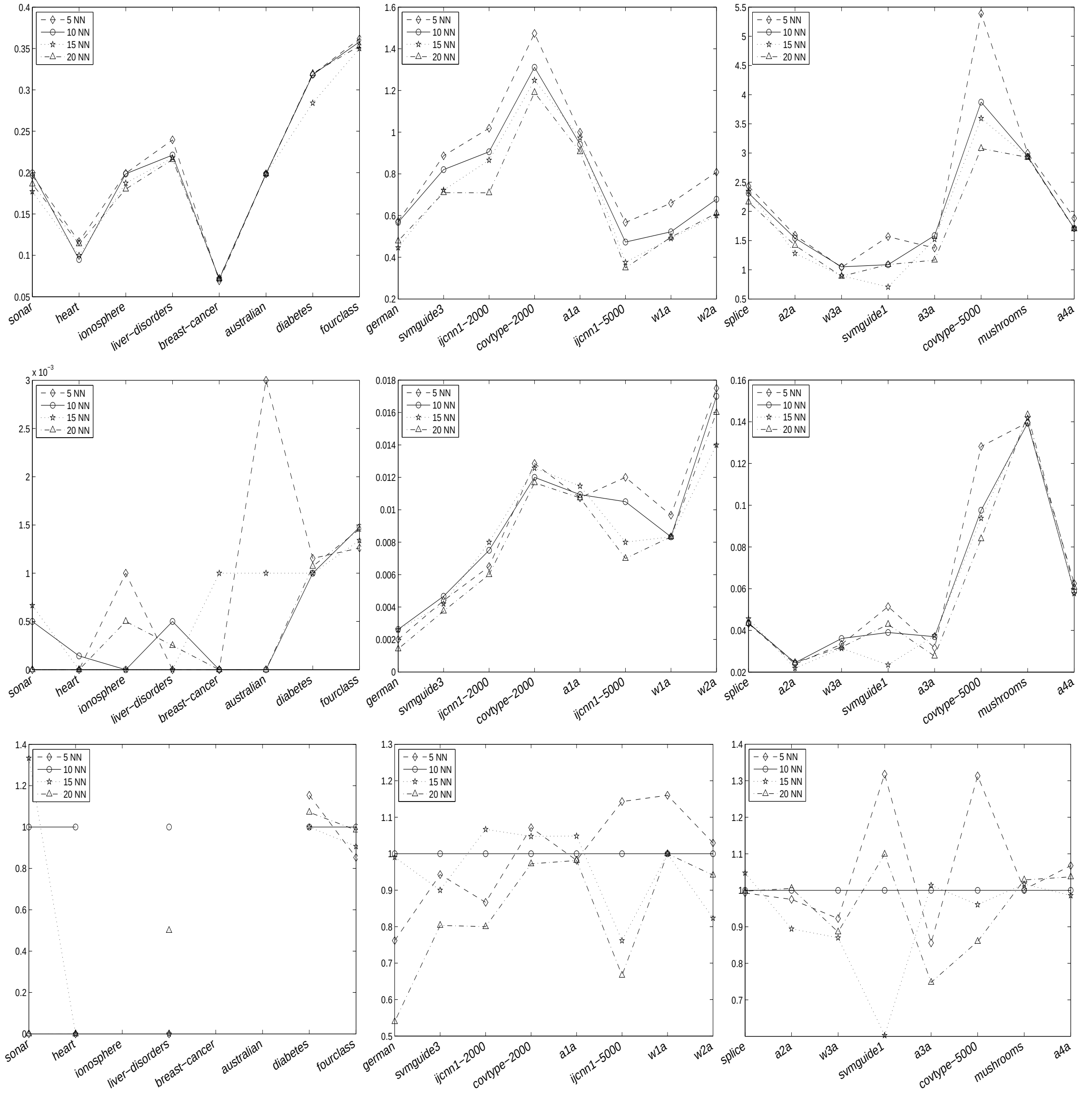

Figure 24: Computational requirements for WSS 7 with different numbers N of nearest neighbors on the grid points whose cross validation error is not larger than 1.05 the minimal cross validation error. The graphics display the average number of iterations in thousands (top), the run time in seconds (middle), and the corresponding ratios $xNN/10NN$ of the run times (bottom). The plots suggest that for grid points with good validation error the number of nearest neighbors has a stronger influence than for the average grid point, yet it is unclear to which extend this effect is caused by different hyper-parameters picked by the different methods.

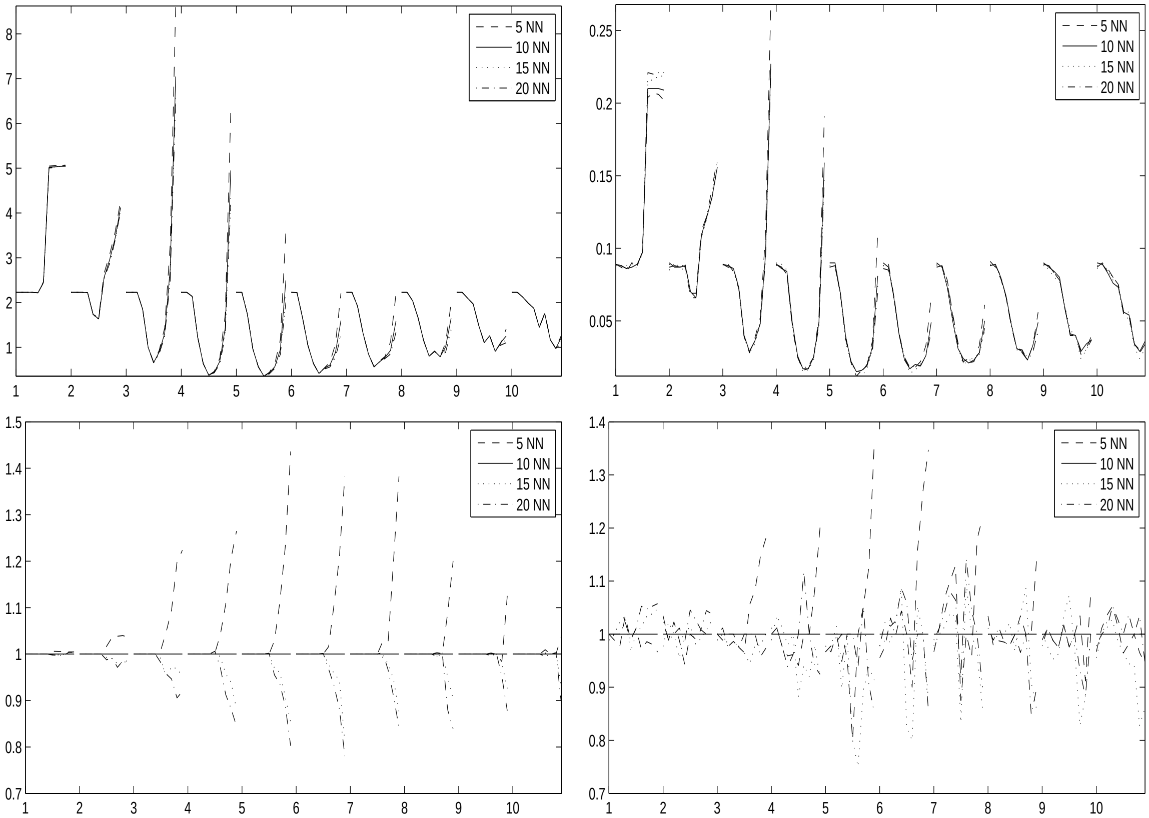

Figure 25: Average computational requirements per grid point of WSS 7 with different numbers N of nearest neighbors for the SVMGUIDE1 data set. The four graphics have the same format as the ones in Figure 10. The graphics at the top display the number of iterations in thousands (left) and the run time in seconds (right), both averaged over the 10 folds, while the graphics at the bottom display the corresponding ratios $xNN/10NN$. Using 5 nearest neighbors clearly results in a worse performance compared to using 10 nearest neighbors. Moreover, compared to $N = 10$ the number of iterations can be further reduced by using more nearest neighbors, but due to unreliable measurements of the run time, it remains somewhat unclear, if this results in significantly shorter run times.

A.4 Results for the Different Initialization Strategies

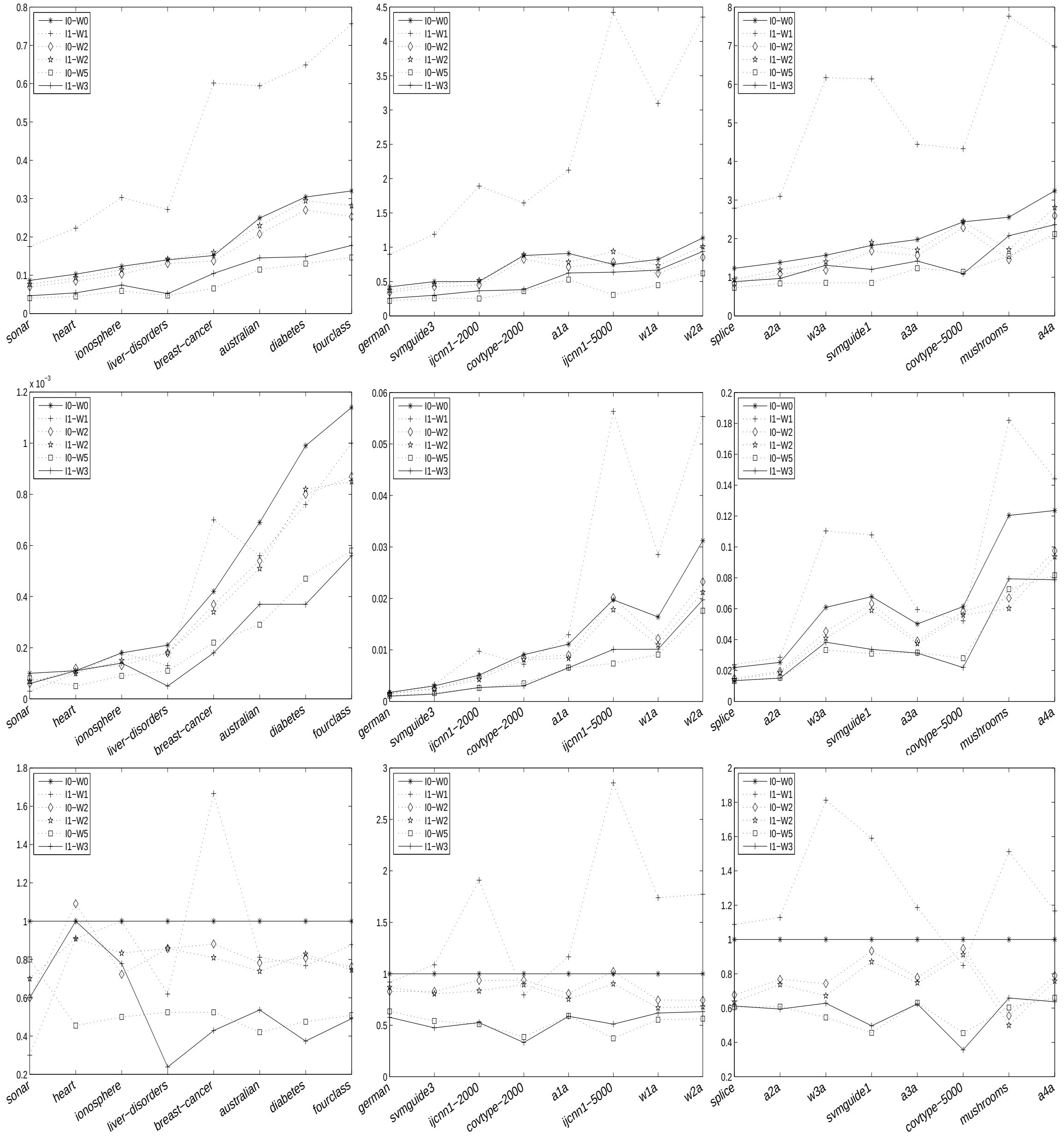

Figure 26: Average computational requirements per grid point of simple initialization strategies for the 2D-SVM with WSS 7. The graphics display the number of iterations in thousands (top), the run time in seconds (middle), and the ratios Ix-Wy/I0-W0 of the run times (bottom).

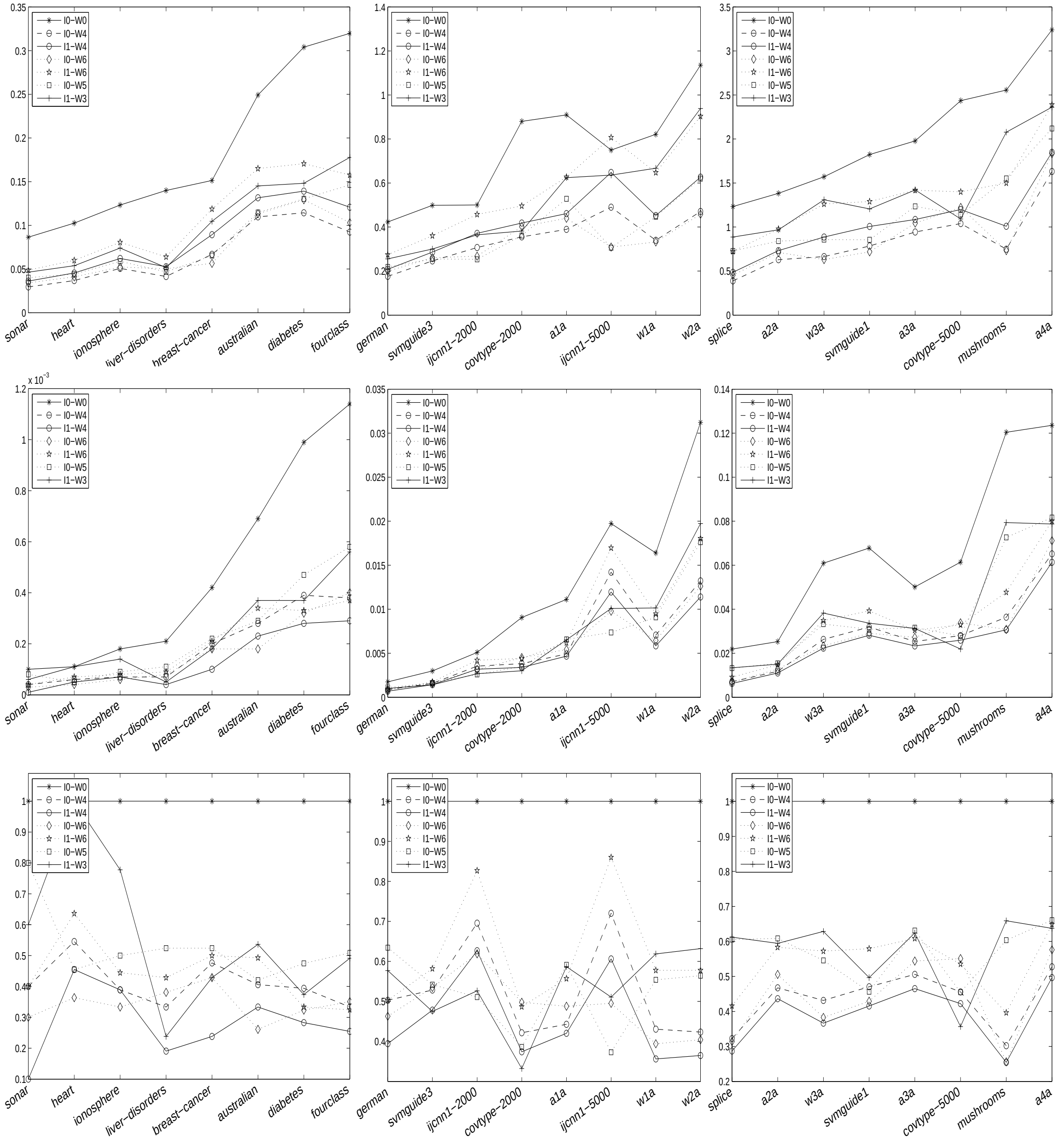

Figure 27: Average computational requirements per grid point of more complex initialization strategies for the 2D-SVM with WSS 7. The graphics display the number of iterations in thousands (top), the run time in seconds (middle), and the ratios Ix-Wy/I0-W0 of the run times (bottom). Note that, again, the cold start initializations with zeros (I0-plots) need less iterations but in most cases more run time. In almost all cases, the more complicated initialization strategies perform better than the simple warm start approaches. Overall, I0-W4, I1-W4, and I0-W6 are the most efficient methods in terms of run time.

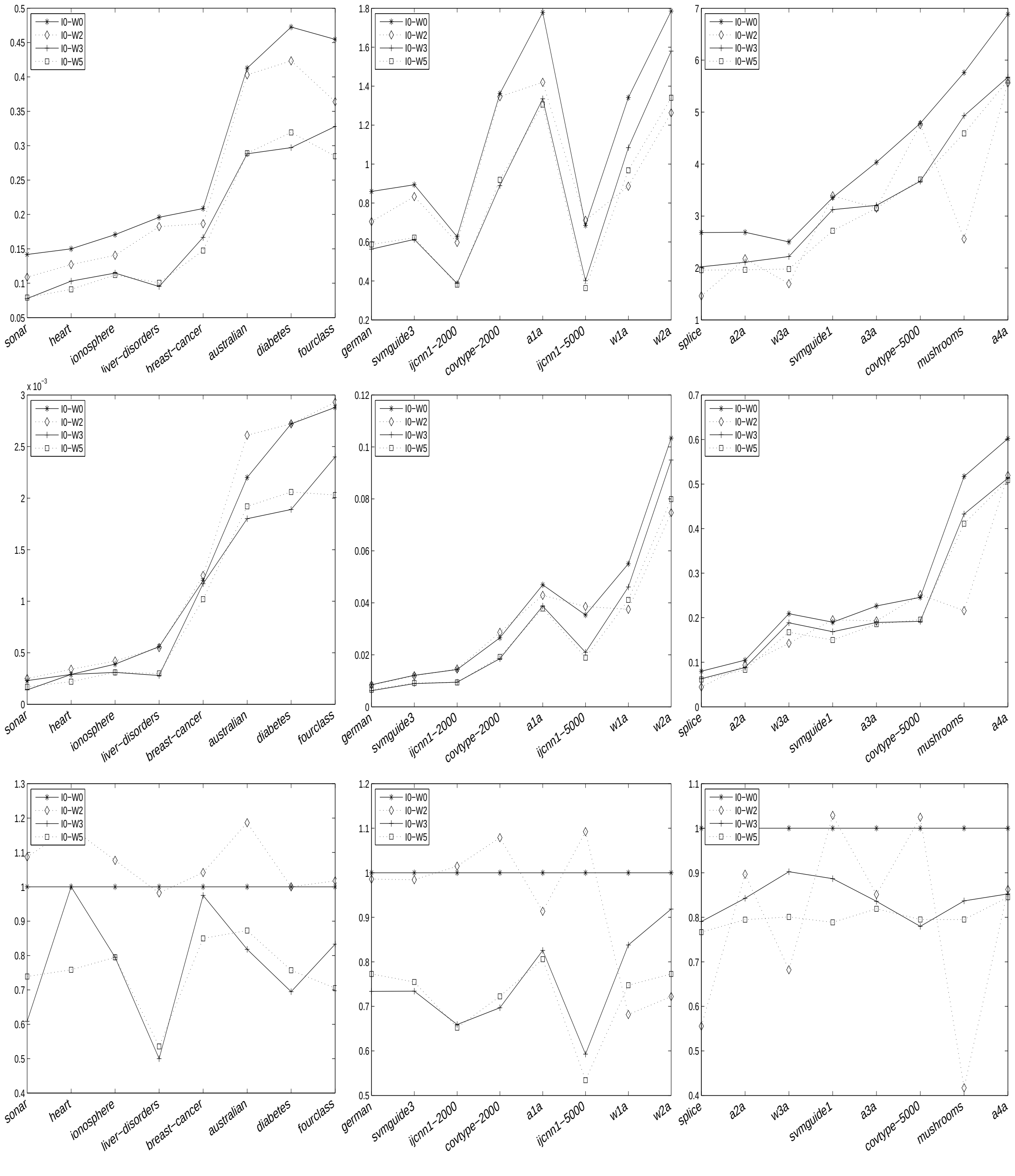

Figure 28: Average computational requirements per grid point of more complex initialization strategies for the LIBSVM for small (left), mid-sized (middle), and relatively large data sets (right). The graphics display the number of iterations in thousands (top), the run time in seconds (middle), and the ratios Ix-Wy/I0-W0 of the run times (bottom). Like for SVMs without offset, using a warm start pays off for this SVM with offset, but the gain is less pronounced.

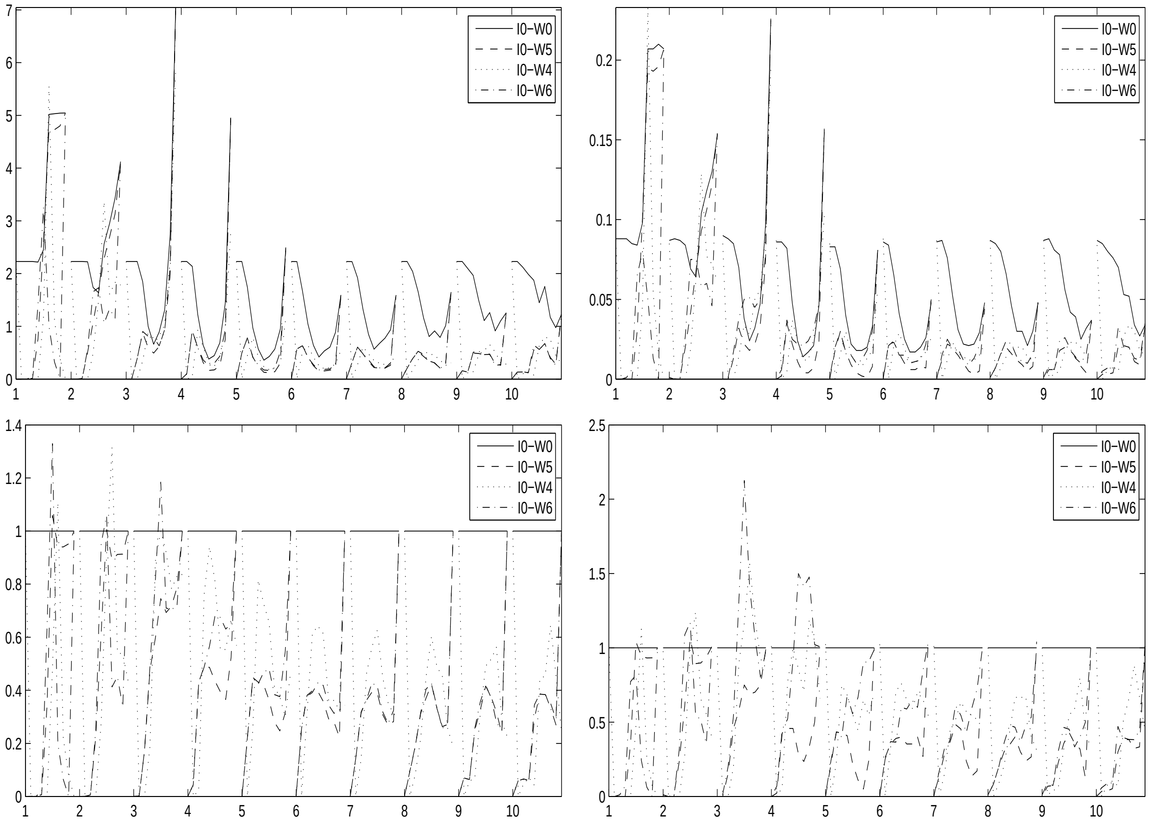

Figure 29: Computational requirements per single grid point of some initialization strategies for the SVMGUIDE1 data set. The four graphics have the same format as the ones in Figure 10. The graphics at the top display the number of iterations in thousands (left) and the run time in seconds (right), both averaged over the 10 folds, while the graphics at the bottom display the corresponding ratios I0-Wx/I0-W0. All warm start strategies perform almost uniformly better than the cold start option I0-W0. Moreover, note that the strategies I0-W5 and I0-W6 start with the smallest λ, that is, at the right hand side of each cell, whereas I0-W4 starts with the largest λ, that is, on the left hand side of each cell.

References

C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines. `http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz`, 2009.

P.-H. Chen, R.-E. Fan, and C.-J. Lin. A study on SMO-type decomposition methods for support vector machines. *IEEE Trans. Neural Networks*, 17:893–908, 2006.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, 2000.

L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, New York, 1996.

R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training support vector machines. *J. Mach. Learn. Res.*, 6:1889–1918, 2005.

G. Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In *KDD ’01: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 77–86, New York, NY, USA, 2001. ACM.

T. Glasmachers and C. Igel. Maximum-gain working set selection for SVMs. *J. Mach. Learn. Res.*, 7:1437–1466, 2006.

C.-W. Hsu and C.-J. Lin. A simple decomposition method for support vector machines. *Mach. Learn.*, 46:291–314, 2002.

C.-W. Hsu and C.-J. Lin. BSVM. `http://www.csie.ntu.edu.tw/~cjlin/bsvm/`, 2006.

T.-M. Huang, V. Kecman, and I. Kopriva. *Kernel Based Algorithms for Mining Huge Data Sets: Supervised, Semi-supervised, and Unsupervised Learning*. Springer, Berlin, 2006.

D. Hush and C. Scovel. Polynomial-time decomposition algorithms for support vector machines. *Mach. Learn.*, 51:51–71, 2003.

D. Hush, P. Kelly, C. Scovel, and I. Steinwart. QP algorithms with guaranteed accuracy and run time for support vector machines. *J. Mach. Learn. Res.*, 7:733–769, 2006.

T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

V. Kecman, T.-M. Huang, and M. Vogt. Iterative single data algorithm for training kernel machines from huge data sets: Theory and performance. In L. Wang, editor, *Support Vector Machines: Theory and Applications*, pages 255–274. Springer Verlag, 2005.

S. Keerthi, V. Sindhwani, and O. Chapelle. An efficient method for gradient-based adaptation of hyperparameters in SVM models. In *Advances in Neural Information Processing Systems 19*, pages 673–680. MIT Press, Cambridge, MA, 2007.

S. S. Keerthi, S. K. Shevade, C. Battacharyya, and K. R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Comput.*, 13:637–649, 2001.

C. J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Trans. Neural Networks*, 12:1288–1298, 2001.

C. J. Lin. Asymptotic convergence of an SMO algorithm without any assumptions. *IEEE Trans. Neural Networks*, 13:248–250, 2002a.

C. J. Lin. A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Trans. Neural Networks*, 13:248–250, 2002b.

N. List and H.-U. Simon. A general convergence theorem for the decomposition method. In *Proceedings of the 17th Annual Conference on Learning Theory*, pages 363–377. Springer, Heidelberg, 2004.

N. List and H. U. Simon. General polynomial time decomposition algorithms. In S. Ben-David, J. Case, and A. Maruko, editors, *Proceedings of the 18th Annual Conference on Learning Theory, COLT 2005*, pages 308–322. Springer, Heidelberg, 2005.

N. List and H. U. Simon. General polynomial time decomposition algorithms. *J. Mach. Learn. Res.*, 8:303–321, 2007.

N. List, D. Hush, C. Scovel, and I. Steinwart. Gaps in support vector optimization. In N. Bshouty and C. Gentile, editors, *Proceedings of the 20th Conference on Learning Theory*, pages 336–348. Springer, New York, 2007.

L.Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *J. Optimization Theory Appl.*, 72:7–35, 1992.

O. L. Mangasarian and D. R. Musicant. Lagrangian support vector machines. *J. Mach. Learn. Res.*, 1:161–177, 2001.

I. Steinwart. Sparseness of support vector machines. *J. Mach. Learn. Res.*, 4:1071–1105, 2003.

I. Steinwart and A. Christmann. *Support Vector Machines*. Springer, New York, 2008.

I. Steinwart, D. Hush, and C. Scovel. An oracle inequality for clipped regularized risk minimizers. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1321–1328. MIT Press, Cambridge, MA, 2007.

M. Vogt. SMO algorithms for support vector machines without bias. Technical report, University of Darmstadt, 2002. `http://www.rtm.tu-darmstadt.de/ehemalige_mitarbeiter/~vogt/docs/vogt_2002_smowob.pdf`.