6 Some Implementation Issues for Algorithm 4

To solve the sub-problem (2.2) by an inner-level CD with a greedy selection, we can consider two implementations.

- At each inner CD step, $\nabla_B f(\alpha)$ is calculated and we use it to find a maximal violating pair from $B$.
- The sub-problem (2.2) is constructed first. Then similar to the situation of training kernel SVM, the gradient $\nabla_B f(\alpha)$ is maintained after each inner update.

We discuss the cost of each implementation in detail. For the first one, the main cost at each inner CD procedure is on calculating $\nabla_B f(\alpha)$ by $|B|$ inner products; see (2.3). In addition, after a maximal violating pair $(i, j)$ has been identified, we need an inner product between $x_i$ and $x_j$ to find $Q_{ij}$ in the two-variable sub-problem (2.11). See details of this implementation in Algorithm 4. Therefore, the total cost of solving a sub-problem is

$$O(|B| + 1)n \times \#\text{inner CD iterations}.$$  

(6.15)

For the second implementation, the major cost is on calculating $Q_{BB}$ and $\nabla_B f(\alpha)$ in the beginning. By taking the symmetry of $Q_{BB}$ into account, the cost for solving the sub-problem is

$$O\left(\left(|B| + \frac{|B|(|B| - 1)}{2}\right)n\right).$$  

(6.16)

Note that for both implementations, once a two-variable sub-problem is formed, the solution procedure at an inner CD step takes only a constant number of operations.

At the first glance the second implementation seems to be faster. If $|B| = 4$ and more than two inner CD steps are taken, (6.15) becomes higher than (6.16). However, in practice we find that the first implementation is competitive because for a small $B$, we may run just one inner CD step.

7 Solution Procedure for a Two-variable Sub-problem

In (2.11) we present the sub-problem in a general form for binary SVM as well as one-class SVM/SVDD; see the use of $y_i, y_j$ in the linear constraint. In software such as LIBSVM, the same code is used for all cases. However, here for the implementation of LIBLINEAR, (2.11) is used only for the one-class situation.1 Therefore, the code becomes simpler. Here we derive some details.

Define

$$a_{ij} = \|x_i\|^2 + \|x_j\|^2 - 2x_i^T x_j$$

$$b_{ij} = -\nabla_i f(\alpha) + \nabla_j f(\alpha)$$

From $d_i = -d_j$, the objective function of (2.7) can be written as

$$\frac{1}{2}a_{ij}d_j^2 + b_{ij}d_j.$$  

If constants are not considered, by minimizing the above quadratic function we can update $\alpha$ as follows.

$$\alpha_{i}^{\text{new}} = \alpha_i + \frac{b_{ij}}{a_{ij}}, \alpha_{j}^{\text{new}} = \alpha_j - \frac{b_{ij}}{a_{ij}}.$$  

(7.17)

We then follow the derivation in [1] to handle the situation if $(\alpha_{i}^{\text{new}}, \alpha_{j}^{\text{new}})$ in (7.17) does not satisfy the bound constraints. Specifically, $(\alpha_{i}^{\text{new}}, \alpha_{j}^{\text{new}})$ must be in one of the four regions outside the following box.

- region I: $\alpha_i + \alpha_j = C_i$
- region II: $\alpha_i + \alpha_j = C_j$
- region III: $\alpha_i = C_i$
- region IV: $\alpha_j = C_j$

1For methods in LIBLINEAR to solve the dual of other problems (classification and regression), SVM without bias is considered. Thus the linear constraint does not appear and CD with $|B| = 1$ is used.
Note that \((α_i^{\text{new}}, α_j^{\text{new}})\) does not appear in the “NA” regions because the current \((α_i, α_j)\) is in the box and
\[
α_i^{\text{new}} + α_j^{\text{new}} = α_i + α_j.
\]
Next, if \((α_i^{\text{new}}, α_j^{\text{new}})\) is outside the box, we identify the region it resides and map it back to the feasible region. If \((α_i^{\text{new}}, α_j^{\text{new}})\) is in region I, we have
\[
α_i^{\text{new}} + α_j^{\text{new}} > C_i \text{ and } α_i^{\text{new}} > C_i.
\]
Then by setting
\[
α_i ← C_i \text{ and } α_j ← (α_i + α_j) - C_i,
\]
the optimal solution of the sub-problem is obtained. Other cases are similar. We have the following pseudo code to identify which region \((α_i^{\text{new}}, α_j^{\text{new}})\) is in and modify \((α_i^{\text{new}}, α_j^{\text{new}})\) to satisfy bound constraints.

```c
double quad_coef = Q_i[i] + Q_j[j] - 2*Q_i[j];
if(quad_coef <= 0)
    quad_coef = 1e-12;
double delta = (G[i]-G[j])/quad_coef;
double sum = alpha[i] + alpha[j];
alpha[i] = alpha[i] - delta;
alpha[j] = alpha[j] + delta;
if(sum > Ci)
{
    if(alpha[i] > Ci) // in region I
    {
        alpha[i] = Ci;
        alpha[j] = sum-Ci;
    }
}
else
{
    if(alpha[j] < 0) // in region III
    {
        alpha[j] = 0;
        alpha[i] = sum;
    }
}
if(sum > Cj)
{
    if(alpha[j] > Cj) // in region II
    {
        alpha[j] = Cj;
        alpha[i] = sum-Cj;
    }
}
else
{
    if(alpha[i] < 0) // in region IV
    {
        alpha[i] = 0;
        alpha[j] = sum;
    }
}
}
```

8 Convergence of Algorithm 5
We faced some difficulties while proving the convergence of Algorithm 5. However, by a simple modification, the asymptotic convergence can be established. In [6], the authors studied the convergence of an optimization framework, where each iteration involves the following two components.
- A two-variable CD sub-problem is solved by using a maximal violating pair.
- If certain conditions hold, then the iterate \(α\) may be further changed to decrease the function value.

The details are in Algorithm 6. In Algorithm 5, we use (3.18) to decide \(r\) pairs, where the first one is the maximal violating pair. Therefore, in the beginning of each outer iteration, Algorithm 5 solves the same two-variable sub-problem as Algorithm 6. Besides using the maximal violating pair, [6] intends to consider a general algorithm regardless of what the rest of an iteration does. However, they fail to prove the convergence for such an algorithm. Instead, they impose a criterion in (8.20) of Algorithm 6 so that only under certain circumstances, additional operations are allowed. These additional operations are arbitrary as long as a simple decreasing condition in (8.21) is satisfied. Now we modify Algorithm 5 to impose the same criterion in (8.20). If we can further prove that solving the rest \(r - 1\) sub-problems leads to a new \(α\) satisfying (8.21), then we have a special case of Algorithm 6 and the convergence holds. To prove (8.21), we define
\[
α^k = α^{k,1}, α^{k,2}, \ldots, α^{k,r}
\]
as the inner iterates and have the following lemma.

**Lemma 8.1.** There exists \(λ > 0\) such that for all \(k = 1, \ldots, r\),
\[
f(α^{k,t+1}) ≤ f(α^{k,t}) - λ||α^{k,t+1} - α^{k,t}||^2,
\]
and
\[
f(α^{k,t+1}) ≤ f(α^k) - λ||α^{k,t+1} - α^k||^2.
\]

**Proof.** From \(α^{k,t}\) to \(α^{k,t+1}\), an inner working set \([i^k,t, j^k,t]\) is considered. If it forms a violating pair at \(α^{k,t}\), [5] has proved (8.18). If it is not a violating pair, from classic results such as Theorem 2 of [3] we have
\[
α^{k,t+1} = α^{k,t}.
\]
Thus, (8.18) holds for any \(λ > 0\).
Algorithm 6 Algorithm analyzed in [6]

1: Let \( \alpha \) be a feasible point, \( \lambda > 0, p \geq 1 \).
2: while stopping condition is not satisfied do
3: Select a maximal violating pair \((i, j)\).
4: Find the optimal solution \( d_B \) for subproblem (2.2) by using \( B = \{i, j\} \).
5: \( \alpha_B \leftarrow \alpha_B + d_B \)
6: if
7: \( \alpha \) is removed such that
8: \( f(\alpha^{new}) - f(\alpha) \leq -\lambda \|\alpha^{new} - \alpha\|^p \).
9: \( \alpha \leftarrow \alpha^{new} \)
10: end if
end while

From (8.18), we have

\[
\begin{align*}
    f(\alpha^{k+1}) & \leq f(\alpha^k) - \lambda \|\alpha^{k+1} - \alpha^k\|^2 \\
    & \leq f(\alpha^k) - \lambda \|\alpha^{k+1} - \alpha^k\|^2 - \lambda \|\alpha^{k+1} - \alpha^k\|^2 \\
    & \leq \lambda \|\alpha^{k+1} - \alpha^k\|^2.
\end{align*}
\]

Therefore, (8.19) is obtained.

In practice, we find that in early iterations, many \( \alpha \) components are easily bounded so that (8.20) does not hold. Then in an iteration we update only the variables of the maximal violating pair. This causes a huge waste on expensively calculating the gradient. Therefore, for the practical use the procedure in Algorithm 5 should be considered.

9 Additional Experimental Results on One-class SVM and SVDD

We present results of one-class SVM with \( \nu = 0.01, 0.005 \) in Figures 4 and 6, and SVDD with \( C = 1/f(l) \) in Figures 5 and 7. The results are similar to Section 4.

To confirm that checking the number of \( O(n) \) operations is similar to checking the running time, in Figures 8 and 9 we present running time results of using \( \nu = 0.1 \). By a comparison with Figure 2, it can be found that the results are similar. However, a careful check shows that the difference between greedy-0.1-cyclic and the other methods are sometimes more dramatic in Figures 8 and 9. For example, considering the set \( \mathbf{a9a} \) and linear one-class SVM with \( \nu = 0.1 \), the number of \( O(n) \) operations for cyclic-2cd to reach \( 10^{-2} \) relative difference is about five times more than that of greedy-0.1-cyclic (see Figure 2). However, in the same problem setting, the time cyclic-2cd spent to reach \( 10^{-2} \) is more than 20 times than that of greedy-0.1-cyclic (see Figure 8(a)). The reason may be that for the random or cyclic working-set selection, easily the sub-problem is optimal. For such sub-problems, some overhead not involving \( O(n) \) operations is not considered in Figure 2. Thus the setting of Figure 2 favors the greedy-0.1-cyclic.

10 Shrinking

Shrinking technique [4] has been well-developed for SVM problems with the bounded constraints \( 0 \leq \alpha_i \leq C_i \). The idea is to tentatively remove some bounded variables to make the optimization process solve a smaller problem and reduce the running time.

We extend the shrinking technique from LIBSVM. For a bound-constrained convex problem like (3.14) or (3.16), we mentioned in Section 2.5 that \( \alpha \) is optimal if and only if (2.7) holds. Let \( \alpha^k \) be the iterate at the beginning of the 4th outer iteration. We define the following two values for each outer iteration to indicate the violation of the optimality condition,

\[
M^k \equiv \max_{i \in I_{up}(\alpha^k)} -y_i \nabla_i f(\alpha^k), \quad m^k \equiv \min_{i \in I_{low}(\alpha^k)} -y_i \nabla_i f(\alpha^k).
\]

Then the variable \( \alpha^k_i \) is removed if one of the following two conditions holds:

\[
l(\alpha^k) \equiv \begin{cases} 
- y_i \nabla_i f(\alpha^k) > M^k \quad \text{if } \alpha^k_i \in I_{up}(\alpha^k), \\
- y_i \nabla_i f(\alpha^k) < m^k \quad \text{if } \alpha^k_i \in I_{low}(\alpha^k).
\end{cases}
\]

We define the following active set to indicate the remained variables for forming a smaller optimization problem.

\[
A \equiv \{ i \mid i \text{ does not satisfy } l(\alpha^k) \}.
\]

We further define

\[
\bar{A} = \{1, \ldots, l\} \setminus A.
\]

For one-class SVM, the new optimization problem is

\[
\min_{\alpha_A} \frac{1}{2} \alpha_A^T Q_{AA} \alpha_A + \alpha_A^T Q_{A} \alpha_A
\]

subject to

\[
0 \leq \alpha_i \leq \frac{1}{p}, \quad i \in A,
\]

\[
e^T_A \alpha_A = 1 - e^T_A \alpha_{\bar{A}}.
\]

Copyright © 2019 Copyright retained by principal author’s organization
The settings of other methods are the same as in Section 4.1. Figures 15-18 present the comparison. Results show that the new setting cyclic-0.1-greedy-0.1-cyclic is useful. Not only is it effectively address the issue of calculating the first whole $\nabla f(\alpha)$ in the beginning, but also the overall convergence is slightly faster. However, the implementation is more complicated and one extra parameter $\bar{R}$ must be decided. Furthermore, how to suitably incorporate the shrinking technique is an issue. Therefore, for simplicity we think greedy-0.1-cyclic can be used if the problem is not extremely large.

References


Algorithm 7  A shrinking implementation of Algorithm 5

1: Given $\epsilon$.
2: Let $\alpha$ be a feasible point.
3: Calculate $Q_{ii}$ and $u = \sum_{i=1}^l \alpha_i x_i, \forall i$.
4: Let $A \leftarrow \{1, \ldots, l\}$.
5: while True do
6:   Let $M \leftarrow -\infty, m \leftarrow \infty$.
7:   for all $i \in A$ do
8:     Calculate the gradient $\nabla_i f(\alpha)$ by (2.3)
9:     if $i \in I_{up}(\alpha)$ and $\nabla_i f(\alpha) > M$ then
10:        $M \leftarrow \nabla_i f(\alpha)$
11:     else if $i \in I_{low}(\alpha)$ and $\nabla_i f(\alpha) < m$ then
12:        $m \leftarrow \nabla_i f(\alpha)$
13:     end if
14:   end for
15:   if $M - m < \epsilon$ then
16:      if $A = \{1, \ldots, l\}$ then
17:         break
18:      else
19:         $A \leftarrow \{1, \ldots, l\}$
20:         continue
21:     end if
22:   end if
23:   for all $i \in A$ do
24:     if $i \notin I_{up}(\alpha)$ and $\nabla_i f(\alpha) > M$ then
25:        $A \leftarrow A \setminus \{i\}$.
26:     else if $i \notin I_{low}(\alpha)$ and $\nabla_i f(\alpha) < m$ then
27:        $A \leftarrow A \setminus \{i\}$.
28:     end if
29:   end for
30:   Find the $r$ most violating pairs by (3.18)
31:     $B = \{i_1, j_1\} \cup \{i_2, j_2\} \cup \cdots \cup \{i_r, j_r\} \subseteq A$
32:   for $s = 1, \ldots, r$ do
33:      $(i, j) \leftarrow B_s$
34:      if $I_{up}(\alpha_{B_s}) = \emptyset$ or $I_{low}(\alpha_{B_s}) = \emptyset$ then
35:         continue
36:      end if
37:      Calculate $\nabla_i f(\alpha)$ and $\nabla_j f(\alpha)$
38:      if $(i, j)$ is not a violating pair then
39:         continue
40:      end if
41:      Calculate $Q_{ij}$ to form the sub-problem (2.11)
42:   end for
43:   end while

Algorithm 8  An extension of Algorithm 5 without calculating the whole gradient

1: Let $\alpha$ be a feasible point and calculate $Q_{ii}, \forall i$
2: while $\alpha$ is not optimal do
3:   Randomly/cyclicly select a large working set $\hat{B}$
4:   Calculate $\nabla_{\hat{B}} f(\alpha)$
5:   Select the $r$ most violating pairs from $\hat{B}$ to have
6:     $B = \{i_1, j_1\} \cup \{i_2, j_2\} \cup \cdots \cup \{i_r, j_r\}$.
7:   Run the for loop in lines 5-16 of Algorithm 5
8: end while

Algorithm 9  A framework considered in Algorithm 5 of [2]

1: Let $\alpha$ be a feasible point and calculate $Q_{ii}, \forall i$
2: while true do
3:   Select a set $\hat{B}$
4:   Calculate $\nabla_{\hat{B}} f(\alpha)$ in parallel
5:   Select $B \subset \hat{B}$ with $|B| \ll |\hat{B}|$
6:   Update $\alpha_B$
7: end while

Copyright © 2019 Copyright retained by principal author’s organization
Figure 4: A comparison on the convergence speed of different methods. Linear one-class SVM with $\nu = 0.01$ is considered. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).

Figure 5: A comparison on the convergence speed of different methods. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.01$ from Figure 4, is used. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).
Figure 6: A comparison on the convergence speed of different methods. Linear one-class SVM with $\nu = 0.005$ is considered. The x-axis is the cumulative number of $O(n)$ operations, while the y-axis is the relative difference to the optimal function value defined in (4.19).

Figure 7: A comparison on the convergence speed of different methods. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.005$ from Figure 6, is used. The x-axis is the cumulative number of $O(n)$ operations, while the y-axis is the relative difference to the optimal function value defined in (4.19).
Figure 8: A comparison on the convergence speed of different methods. Linear one-class SVM with $\nu = 0.1$ is considered. The $x$-axis is the running time in seconds, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).

Figure 9: A comparison on the convergence speed of different methods. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.1$ from Figure 8, is used. The $x$-axis is the running time in seconds, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).
Figure 10: A comparison on the convergence speed of Algorithm 5 with/without shrinking technique. Linear one-class SVM with $\nu = 0.1$ is considered. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).

Figure 11: A comparison on the convergence speed of Algorithm 5 with/without shrinking technique. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.1$ from Figure 10, is used. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).
Figure 12: A comparison on the convergence speed of Algorithm 5 with/without shrinking technique. Linear one-class SVM with $\nu = 0.01$ is considered. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).

Figure 13: A comparison on the convergence speed of Algorithm 5 with/without shrinking technique. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.01$ from Figure 12, is used. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).
1: Let $\alpha$ be a feasible point and calculate $Q_{ii}, \forall i$

2: while $\alpha$ is not optimal do

3: Select $\bar{B}$

4: Calculate $\nabla_{\bar{B}} f(\alpha)$

5: Select the $r$ most violating indices as $B$

6: $B = \{i_1, \ldots, i_r\}$

7: for $s = 1, \ldots, r$ do

8: Update $\alpha_{i_s}$

9: end for

10: end while

(a) Algorithm 5 in [2].

(b) A simplified description of Algorithm 8.

Figure 14: A line-by-line comparison between an algorithm in [2] and our Algorithm 8
Figure 15: A comparison on the convergence speed of Algorithm 8 and other methods. Linear one-class SVM with $\nu = 0.1$ is considered. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).

Figure 16: A comparison on the convergence speed of Algorithm 8 and other methods. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.1$ from Figure 15, is used. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).
Figure 17: A comparison on the convergence speed of Algorithm 8 and other methods. Linear one-class SVM with $\nu = 0.01$ is considered. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).

Figure 18: A comparison on the convergence speed of Algorithm 8 and other methods. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.01$ from Figure 17, is used. The $x$-axis is the cumulative number of $O(n)$ operations, while the $y$-axis is the relative difference to the optimal function value defined in (4.19).