Supplementary Materials for Dual Coordinate-Descent Methods for Linear One-Class SVM and SVDD

Hung-Yi Chou* Pin-Yen Lin* Chih-Jen Lin*

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, $\nabla_B f(\boldsymbol{\alpha})$ 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 $\nabla_B f(\boldsymbol{\alpha})$ 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 $\nabla_B f(\boldsymbol{\alpha})$ 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 $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ to find Q_{ij} in the two-variable sub-sub-problem (2.11). See details of this implementation in Algorithm 4. Therefore, the total cost of solving a sub-problem is

$$O((|B|+1)n) \times \#\text{inner CD iterations}. \tag{6.15}$$

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

$$O\left(\left(|B| + \frac{|B|(|B|-1)}{2}\right) n\right). \tag{6.16}$$

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.

*Computer Science and Information Engineering, National Taiwan Univ.

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

$$a_{ij} = \|\boldsymbol{x}_i\|_2^2 + \|\boldsymbol{x}_j\|_2^2 - 2\boldsymbol{x}_i^T \boldsymbol{x}_j$$
$$b_{ij} = -\nabla_i f(\boldsymbol{\alpha}) + \nabla_j f(\boldsymbol{\alpha})$$

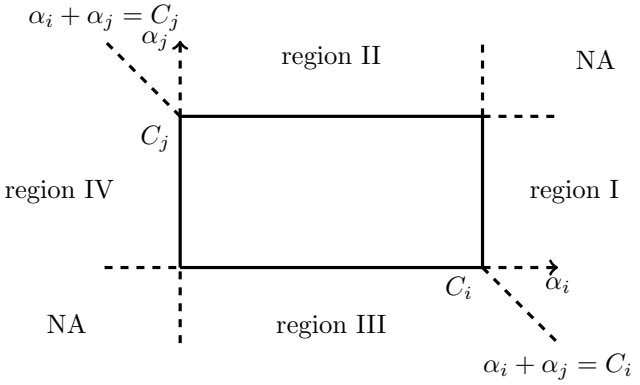
From $d_i = -d_j$, the objective function of (2.7) can be written as

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

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

$$\alpha_i^{\text{new}} = \alpha_i + \frac{b_{ij}}{a_{ij}}, \alpha_j^{\text{new}} = \alpha_j - \frac{b_{ij}}{a_{ij}}. \tag{7.17}$$

We then follow the derivation in [2] 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.



[1] For 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 $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ does not appear in the "NA" regions because the current (α_i, α_j) is in the box and

$$\alpha_i^{\text{new}} + \alpha_j^{\text{new}} = \alpha_i + \alpha_j.$$

Next, if $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ is outside the box, we identify the region it resides and map it back to the feasible region. If $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ is in region I, we have

$$\alpha_i^{\text{new}} + \alpha_j^{\text{new}} > C_i \text{ and } \alpha_i^{\text{new}} > C_i.$$

Then by setting

$$\alpha_i \leftarrow C_i \text{ and } \alpha_j \leftarrow (\alpha_i + \alpha_j) - C_i,$$

the optimal solution of the sub-problem is obtained. Other cases are similar. We have the following pseudo code to identify which region $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ is in and modify $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ to satisfy bound constraints.

```
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 [10], 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 $\boldsymbol{\alpha}$ 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, [10] 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 $\boldsymbol{\alpha}$ satisfying (8.21), then we have a special case of Algorithm 6 and the convergence holds. To prove (8.21), we define

$$\boldsymbol{\alpha}^k = \boldsymbol{\alpha}^{k,1}, \boldsymbol{\alpha}^{k,2}, \ldots, \boldsymbol{\alpha}^{k,r}$$

as the inner iterates and have the following lemma.

LEMMA 8.1. *There exists $\lambda > 0$ such that for all $k = 1, \ldots$ and $t = 1, \ldots, r$,*

$$f(\boldsymbol{\alpha}^{k,t+1}) \le f(\boldsymbol{\alpha}^{k,t}) - \lambda ||\boldsymbol{\alpha}^{k,t+1} - \boldsymbol{\alpha}^{k,t}||^2, \tag{8.18}$$

and

$$f(\boldsymbol{\alpha}^{k,t+1}) \le f(\boldsymbol{\alpha}^{k}) - \lambda ||\boldsymbol{\alpha}^{k,t+1} - \boldsymbol{\alpha}^{k}||^2. \tag{8.19}$$

Proof. From $\boldsymbol{\alpha}^{k,t}$ to $\boldsymbol{\alpha}^{k,t+1}$, an inner working set $\{i_t^k, j_t^k\}$ is considered. If it forms a violating pair at $\boldsymbol{\alpha}^{k,t}$, [9] has proved (8.18). If it is not a violating pair, from classic results such as Theorem 2 of [7] we have $\boldsymbol{\alpha}^{k,t+1} = \boldsymbol{\alpha}^{k,t}$. Thus, (8.18) holds for any $\lambda > 0$.

Algorithm 6 Algorithm analyzed in [10]

Let $\boldsymbol{\alpha}$ be a feasible point, $\lambda > 0$, $p \geq 1$.
while stopping condition is not satisfied **do**
  Select a maximal violating pair (i, j)
  Find the optimal solution $\boldsymbol{d}_B$ for sub-problem (2.2) by using $B = \{i, j\}$.
  $\boldsymbol{\alpha}_B \leftarrow \boldsymbol{\alpha}_B + \boldsymbol{d}_B$
  if

$$0 < \alpha_i < C_i \text{ and } 0 < \alpha_j < C_j \tag{8.20}$$

  then
    find $\boldsymbol{\alpha}^{\text{new}}$ such that

$$f(\boldsymbol{\alpha}^{\text{new}}) - f(\boldsymbol{\alpha}) \leq -\lambda \|\boldsymbol{\alpha}^{\text{new}} - \boldsymbol{\alpha}\|^p. \tag{8.21}$$

    $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}^{\text{new}}$
  end if
end while

From (8.18), we have

$$\begin{aligned}
&f(\boldsymbol{\alpha}^{k,t+1}) \\
\leq& f(\boldsymbol{\alpha}^{k,t}) - \lambda||\boldsymbol{\alpha}^{k,t+1} - \boldsymbol{\alpha}^{k,t}||^2 \\
\leq& f(\boldsymbol{\alpha}^{k,t-1}) - \lambda||\boldsymbol{\alpha}^{k,t} - \boldsymbol{\alpha}^{k,t-1}||^2 - \lambda||\boldsymbol{\alpha}^{k,t+1} - \boldsymbol{\alpha}^{k,t}||^2 \\
\leq& \cdots \\
\leq& f(\boldsymbol{\alpha}^{k}) - \lambda\left(||\boldsymbol{\alpha}^{k,2} - \boldsymbol{\alpha}^{k}||^2 + \cdots + ||\boldsymbol{\alpha}^{k,t+1} - \boldsymbol{\alpha}^{k,t}||^2\right) \\
\leq& f(\boldsymbol{\alpha}^{k}) - \lambda||\boldsymbol{\alpha}^{k,t+1} - \boldsymbol{\alpha}^{k}||^2.
\end{aligned}$$

Therefore, (8.19) is obtained. □

In practice, we find that in early iterations, many $\boldsymbol{\alpha}$ components are easily bounded so that (8.20) does not hold. Then in an iteration we update only the two 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 5 and 7, and SVDD with $C = 1/(\nu l)$ in Figures 6 and 8. 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 9 and 10 we present running time results of using $\nu = 0.1$. By a comparison with Figrue 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 9 and 10. For example, considering the set 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 9(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 cyclic-2cd more than greedy-0.1-cyclic.

10 Shrinking

Shrinking technique [8] 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 in the optimization process to 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 $\boldsymbol{\alpha}$ is optimal if and only if (2.7) holds. Let $\boldsymbol{\alpha}^k$ be the iterate at the beginning of the kth outer iteration. We define the following two values for each outer iteration to indicate the violation of the optimality condition,

$$\begin{aligned}
M^k &\equiv \max_{t \in I_{\text{up}}(\boldsymbol{\alpha}^k)} -y_t \nabla_t f(\boldsymbol{\alpha}^k), \text{ and} \\
m^k &\equiv \min_{t \in I_{\text{low}}(\boldsymbol{\alpha}^k)} -y_t \nabla_t f(\boldsymbol{\alpha}^k).
\end{aligned}$$

Then the variable α_i^k is removed if one of the following two conditions holds:

$$\begin{cases} -y_i \nabla_i f(\boldsymbol{\alpha}^k) > M^k & \text{if } \alpha_i^k \in I_{\text{low}}(\boldsymbol{\alpha}^k), \\ -y_i \nabla_i f(\boldsymbol{\alpha}^k) < m^k & \text{if } \alpha_i^k \in I_{\text{up}}(\boldsymbol{\alpha}^k). \end{cases} \tag{10.22}$$

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

$$A \equiv \{i \mid i \text{ does not satisfy (10.22)}\}.$$

We further define

$$\bar{A} = \{1, \ldots, l\} \backslash A.$$

For one-class SVM, the new optimization problem is

$$\begin{aligned}
\min_{\boldsymbol{\alpha}_A} \quad & \frac{1}{2}\boldsymbol{\alpha}_A^T Q_{AA} \boldsymbol{\alpha}_A + \boldsymbol{\alpha}_{\bar{A}}^T Q_{\bar{A}A} \boldsymbol{\alpha}_A \\
\text{subject to} \quad & 0 \leq \alpha_i \leq \frac{1}{\nu l}, i \in A, \\
& \boldsymbol{e}_A^T \boldsymbol{\alpha}_A = 1 - \boldsymbol{e}_{\bar{A}}^T \boldsymbol{\alpha}_{\bar{A}}.
\end{aligned} \tag{10.23}$$

The gradient of the objective function is

$$Q_{A,:}\begin{bmatrix}\boldsymbol{\alpha}_A\\ \boldsymbol{\alpha}_{\bar{A}}\end{bmatrix}.$$

Because $\boldsymbol{\alpha}_{\bar{A}}$ remained the same and we keep maintaining the vector $\boldsymbol{u} = \sum_{i=1}^{l}\alpha_i\boldsymbol{x}_i$, even for the smaller problem (10.23), the gradient calculation by (2.3) is still correct. Algorithm 7 summarizes the shrinking implementation of the two-level CD with an outer greedy selection and an inner random/cyclic selection (i.e., Algorithm 5).

The comparisons between with and without the shrinking implementations are presented in Figures 11-14. We can see that greedy-0.1-cyclic-shrink generally outperforms greedy-0.1-cyclic. This result shows that the shrinking technique can improve the convergence.

11 An Extension of Algorithm 5

The discussion in Section 4 indicates that a disadvantage of Alglrithm 5 is the expensive calculation of the whole gradient $\nabla f(\boldsymbol{\alpha})$. In particular, $\boldsymbol{\alpha}$ cannot be updated until the first gradient has been obtained. We can relax the full-gradient requirement by considering an extension of Algorithm 5 in Algorithm 8. The basic idea is that at each outer iteration, instead of using $\nabla f(\boldsymbol{\alpha})$, we consider a large set $\bar{B}$ and select r pairs from this set. Thus only $\nabla_{\bar{B}} f(\boldsymbol{\alpha})$ must be calculated.

Interestingly, Algorithm 8 is a special case of a framework considered in [3], which is presented in Algorithm 9 as a comparison. Their algorithm was designed to run coordinate descent methods in a multicore environment. To parallelize operations, their ideas are as follows

1. A large set $\bar{B}$ is selected and $\nabla_{\bar{B}} f(\boldsymbol{\alpha})$ can be calculated in parallel.
2. From $\bar{B}$ a subset B is selected. Then CD steps are sequentially conducted on elements in B.

Naturally, a reasonable selection of B should include elements that can likely be updated. To this end, they select some most violating "indices." Note that they consider SVM without the bias term for binary classification, and use CD to solve the SVM dual. Thus there is no linear constraint and they update one element at each CD step. Their realization of the framework in Algorithm 9 is Algorithm 5 in their paper. By a line by line comparison in Figure 15 we can easily see that theirs and ours are very related.

We conduct experiments to compare Algorithm 5 and Algorithm 8. In Algorithm 8, we consider a cyclic working-set selection in the first-level CD iteration (line 3 in Algorithm 8) and select $\bar{B}$ with size $\bar{R}l$. We label this method as cyclic-$\bar{R}$-greedy-R-cyclic, and consider

$$\bar{R} = 0.1 \text{ and } R = 0.1.$$

The settings of other methods are the same as in Section 4.1. Figures 16-19 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 $\nabla f(\boldsymbol{\alpha})$ in the begining, 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.

12 Speedup Finding the r Most Violating Pairs in Algorithm 7

Motivated by an idea from Hung-Yi Chou, this section is written by Guan-Ting Chen after the paper was published.

We analyze the time complexity of each iteration in Algorithm 7. In each iteration, we conduct the following operations.

- Calculate $\nabla_A f(\boldsymbol{\alpha})$ in $O(|A|n)$, where A is the active set.
- Find the r most violating pairs to form the working set B. We discuss more details in the rest of this section.
- Solve the sub-problem (2.11) in $O(|B|n)$.

To find the r most violating pairs, before version 2.46, the LIBLINEAR software follows [4] to use heap sort. In this implemtation, through checking each $i \in A$ we maintain a max heap and a min heap in size r to reserve the r most violating pairs. Therefore, the cost of constructing a heap is $O(|A|\log|A|)$. The total cost at each iteration is,

$$O((|A|+|B|)n + |A|\log|A|). \tag{12.24}$$

From (12.24), we can see that the time complexity of finding the r most violating pairs is not negligible. The time cost of this part can be more significant if the number of features is small or the data set is sparse, where n in (12.24) is changed to the number of nonzero features per instance. This situation can be seen in Table 1, where we show the percentage of running time on finding the r most violating pairs.

To reduce the cost of finding the r most violating pairs, we investigate the following procedure by applying the quickselect algorithm [6] first.

- A quickselect procedure is conducted to split $-y_t\nabla_t f(\boldsymbol{\alpha})$, $t \in I_{\text{up}}$ to two parts, where one part contains the largest r elements.
  A quickselect procedure is conducted to split $-y_t\nabla_t f(\boldsymbol{\alpha})$, $t \in I_{\text{low}}$ to two parts, where one part contains the smallest r elements.
- For each of the two r-element sets obtained above,

datasets	ijcnn1	a9a	real-sim	rcv1	news20
# of features	22	123	20,958	47,236	1,355,191
average # of non-zero features per instance	13.0	13.87	51.29	74.05	454.99
finding the r most violating pairs	47.66%	35.42%	11.21%	7.67%	0.84%

Table 1: Percentage of running time at each iteration on finding the r most violating pairs in different data sets.

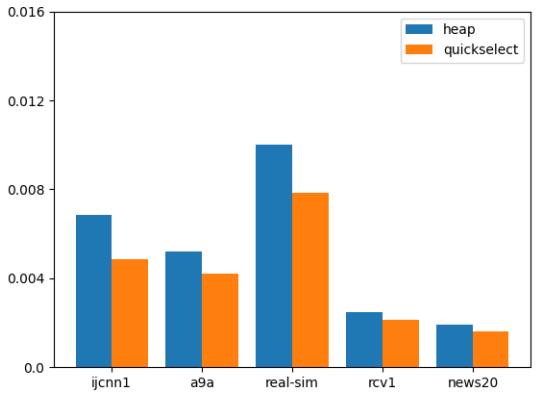


Figure 4: A timing comparsion between the two strategies (heap and quickselect/quicksort) for finding the r most violating pairs. We sum up the running time of this operation in all iterations. The y-axis (in seconds) is the average of five runs.

we sort the elements by quicksort. Then the r most violating pairs defined in (3.18) are found.

The cost of a quickselect algorithm is known to be linear to the input size. With $r = |B|$, the cost of each iteration in Algorithm 7 is

$$O((|A| + |B|)n + |A| + |B| \log |B|). \tag{12.25}$$

A comparsion between (12.24) and (12.25) shows that this new setting can potentially reduce the running time. For the same five data sets considered in Table 1, we compare the two approaches in Figure 4 by showing their total running time on finding the r most violating pairs. The new approach by quickselect is more efficent.

While we have successfully reduced the running time on finding the working set, the operation is generally not the dominant one in each iteration. Instead, $O(|A|n)$ is often the bottleneck in (12.24). Thus, it is important to check the overall time reduction by applying the new strategy of finding the r most violateing pairs. We present the comparison results in Table 2. For problems where finding violating pairs takes a considerable portion of the total running time, the new strategy significantly improves the overall training efficiency.

Next we discuss some implementation details. For quickselect, we consider the following two ways.

- Quickselect with Lomuto partition scheme [1].
- Quickselect with Hoare's partition scheme [5].

A comparison (details not shown) indicates similar running time, so we decide to use the Lomuto partition scheme for better readibility.

We note one implementation detail in our experiments. To ensure a fair comparison, the selected r most violating pairs must be the same. Then the whole optimization algorithm uses the same number of iterations. To fullfill this requirement, we impose a unique order of all elements in applying quickselect or sorting algorithms by defining that

(12.26)
$$i \text{ is ahead of } j \text{ if } \nabla_i f(\boldsymbol{\alpha}) = \nabla_j f(\boldsymbol{\alpha}) \text{ and } i < j.$$

The heap implementation in LIBLINEAR 2.45 does not satisfy (12.26), so we make proper changes for the experiments.

We mentioned that the dominant cost at each iteration is usually the $O(|A|n)$ operations on calculating

$$\nabla_i f(\boldsymbol{\alpha}) = y_i \boldsymbol{u}^T \boldsymbol{x}_i - 1, \forall i \in A. \tag{12.27}$$

The computation is independent under each i, so can be easily parallelized. We check the effect of parallizing (12.27) in Table 3. Because (12.27) dominates the training process, if we redo the comparison in Table 2, the time reduction of using quickselect instead of heap becomes more significant, as shown in Table 3.

We conduct all experiments in this section on a machine with 8 cores of Intel i7-6900K CPUs with 256KiB L1i-cache, 256KiB L1d-cache, 2MiB L2-cache and 20MiB shared L3-cache. The new working-set selection by using quickselect is incorporated in LIBLINEAR (after version 2.46). The code for experiments is

datasets	ijcnn1	a9a	real-sim	rcv1	news20
total runtime using heap	0.0139	0.0136	0.0841	0.0299	0.2117
total runtime using quickselect	0.0120	0.0125	0.0814	0.0295	0.2107
runtime reduction with applying quickselect	13.60%	7.79%	3.12%	1.18%	0.46%

Table 2: The total running time (in seconds; average of five runs) of the one-class SVM solver. We apply two strategies to find the r most violating pairs.

available at `https://www.csie.ntu.edu.tw/~cjlin/papers/linear_oneclass_SVM/quickselect.tar.gz`

References

[1] J. Bentley, *Programming pearls*, Addison-Wesley Professional, second ed., 2016.

[2] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, ACM Transactions on Intelligent Systems and Technology, 2 (2011), pp. 27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] W.-L. Chiang, M.-C. Lee, and C.-J. Lin, *Parallel dual coordinate descent method for large-scale linear classification in multi-core environments*, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2016, `http://www.csie.ntu.edu.tw/~cjlin/papers/multicore_cddual.pdf`.

[4] H.-Y. Chou, P.-Y. Lin, and C.-J. Lin, *Dual coordinate-descent methods for linear one-class SVM and SVDD*, in Proceedings of SIAM International Conference on Data Mining (SDM), 2020, `http://www.csie.ntu.edu.tw/~cjlin/papers/linear_oneclass_SVM/siam.pdf`.

[5] C. A. R. Hoare, *Algorithm 63: partition*, Communications of the ACM, 4 (1961), p. 321.

[6] C. A. R. Hoare, *Algorithm 65: find*, Communications of the ACM, 4 (1961), pp. 321–322.

[7] D. Hush and C. Scovel, *Polynomial-time decomposition algorithms for support vector machines*, Machine Learning, 51 (2003), pp. 51–71, `http://www.c3.lanl.gov/~dhush/machine_learning/svm_decomp.ps`.

[8] T. Joachims, *Making large-scale SVM learning practical*, in Advances in Kernel Methods – Support Vector Learning, B. Schölkopf, C. J. C. Burges, and A. J. Smola, eds., Cambridge, MA, 1998, MIT Press, pp. 169–184.

[9] C.-J. Lin, *Asymptotic convergence of an SMO algorithm without any assumptions*, IEEE Transactions on Neural Networks, 13 (2002), pp. 248–250, `http://www.csie.ntu.edu.tw/~cjlin/papers/q2conv.pdf`.

[10] S. Lucidi, L. Palagi, A. Risi, and M. Sciandrone, *A convergent hybrid decomposition algorithm model for SVM training*, IEEE Transactions on Neural Networks, 20 (2009), pp. 1055–1060.

datasets	ijcnn1	a9a	real-sim	rcv1	news20
Using quickselect in the working set selection					
running (12.27) without parallelization	0.0120	0.0125	0.0814	0.0295	0.2107
running (12.27) by 4 threads	0.0117	0.0116	0.0587	0.0240	0.1573
running (12.27) by 8 threads	0.0119	0.0109	0.0521	0.0213	0.1330
running (12.27) by 12 threads	0.0128	0.0126	0.0543	0.0230	0.1368
runtime reduction by 4 threads	2.88%	7.60%	27.93%	18.58%	25.34%
runtime reduction by 8 threads	10.48%	13.10%	35.98%	27.86%	36.88%
runtime reduction by 12 threads	-7.01%	-0.56%	33.29%	22.18%	35.06%
Using heap in the working set selection					
running (12.27) without parallelization	0.0139	0.0136	0.0841	0.0299	0.2117
running (12.27) by 8 threads	0.0127	0.0120	0.0541	0.0216	0.1352
Total runtime reduction after switching from heap to quickselect					
runtime reduction without parllelization	13.60%	7.79%	3.12%	1.18%	0.46%
runtime reduction by 8 threads	15.41%	9.58%	3.68%	1.26%	1.61%

Table 3: The total running time (in seconds) with/without parallelizing the operation in (12.27). Time for data loading is excluded.

Algorithm 7 A shrinking implementation of Algorithm 5

Given ϵ.
Let $\boldsymbol{\alpha}$ be a feasible point.
Calculate Q_{ii} and $\boldsymbol{u} = \sum_{i=1}^{l} \alpha_i \boldsymbol{x}_i, \forall i$.
Let $A \leftarrow \{1, \ldots, l\}$.
while True **do**
    Let $M \leftarrow -\infty, m \leftarrow \infty$.
    for all $i \in A$ **do**
        Calculate the gradient $\nabla_i f(\boldsymbol{\alpha})$ by (2.3)
        if $i \in I_{\text{up}}(\boldsymbol{\alpha})$ and $\nabla_i f(\boldsymbol{\alpha}) > M$ **then**
            $M \leftarrow \nabla_i f(\boldsymbol{\alpha})$
        else if $i \in I_{\text{low}}(\boldsymbol{\alpha})$ and $\nabla_i f(\boldsymbol{\alpha}) < m$ **then**
            $m \leftarrow \nabla_i f(\boldsymbol{\alpha})$
        end if
    end for
    if $M - m < \epsilon$ **then**
        if $A = \{1, \ldots, l\}$ **then**
            break
        else
            $A \leftarrow \{1, \ldots, l\}$
            continue
        end if
    end if
    for all $i \in A$ **do**
        if $i \notin I_{\text{up}}(\boldsymbol{\alpha})$ and $\nabla_i f(\boldsymbol{\alpha}) > M$ **then**
            $A \leftarrow A \backslash \{i\}$.
        else if $i \notin I_{\text{low}}(\boldsymbol{\alpha})$ and $\nabla_i f(\boldsymbol{\alpha}) < m$ **then**
            $A \leftarrow A \backslash \{i\}$.
        end if
    end for
    Find the r most violating pairs by (3.18)

$$B = \underbrace{\{i_1, j_1\}}_{B_1} \cup \cdots \cup \underbrace{\{i_r, j_r\}}_{B_r} \subseteq A$$

    for $s = 1, \ldots, r$ **do**
        $(i, j) \leftarrow B_s$
        if $I_{\text{up}}(\boldsymbol{\alpha}_{B_s}) = \emptyset$ or $I_{\text{low}}(\boldsymbol{\alpha}_{B_s}) = \emptyset$ **then**
            continue
        end if
        Calculate $\nabla_i f(\boldsymbol{\alpha})$ and $\nabla_j f(\boldsymbol{\alpha})$
        if (i, j) is not a violating pair **then**
            continue
        end if
        Calculate Q_{ij} to form the sub-problem (2.11)
        Solve the sub-problem.
        Update (α_i, α_j) and the vector $\boldsymbol{u}$ by (2.5).
    end for
end while

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

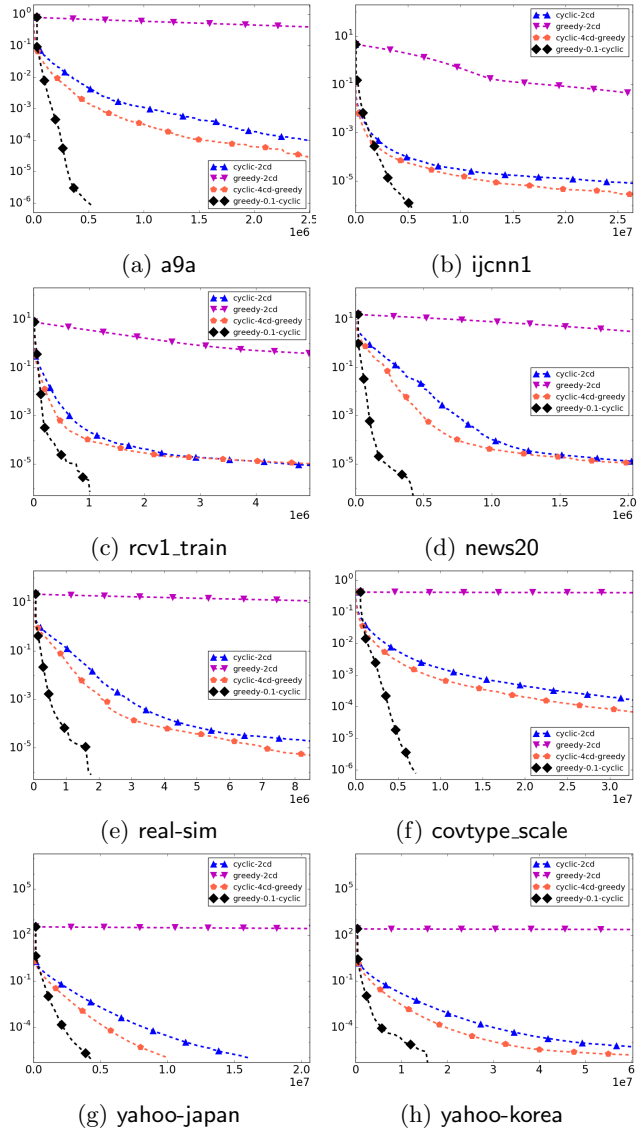
Let $\boldsymbol{\alpha}$ be a feasible point and calculate $Q_{ii}, \forall i$
while $\boldsymbol{\alpha}$ is not optimal **do**
    Randomly/cyclicly select a large working set $\bar{B}$
    Calculate $\nabla_{\bar{B}} f(\boldsymbol{\alpha})$
    Select the r most violating pairs from $\bar{B}$ to have

$$B = \{i_1, j_1\} \cup \{i_2, j_2\} \cup \cdots \cup \{i_r, j_r\}.$$

    Run the **for** loop in lines 5-16 of Algorithm 5
end while

Algorithm 9 A framework considered in Algorithm 5 of [3]

Let $\boldsymbol{\alpha}$ be a feasible point and calculate $Q_{ii}, \forall i$
while true **do**
    Select a set $\bar{B}$
    Calculate $\nabla_{\bar{B}} f(\boldsymbol{\alpha})$ in parallel
    Select $B \subset \bar{B}$ with $|B| \ll |\bar{B}|$
    Update $\boldsymbol{\alpha}_B$
end while



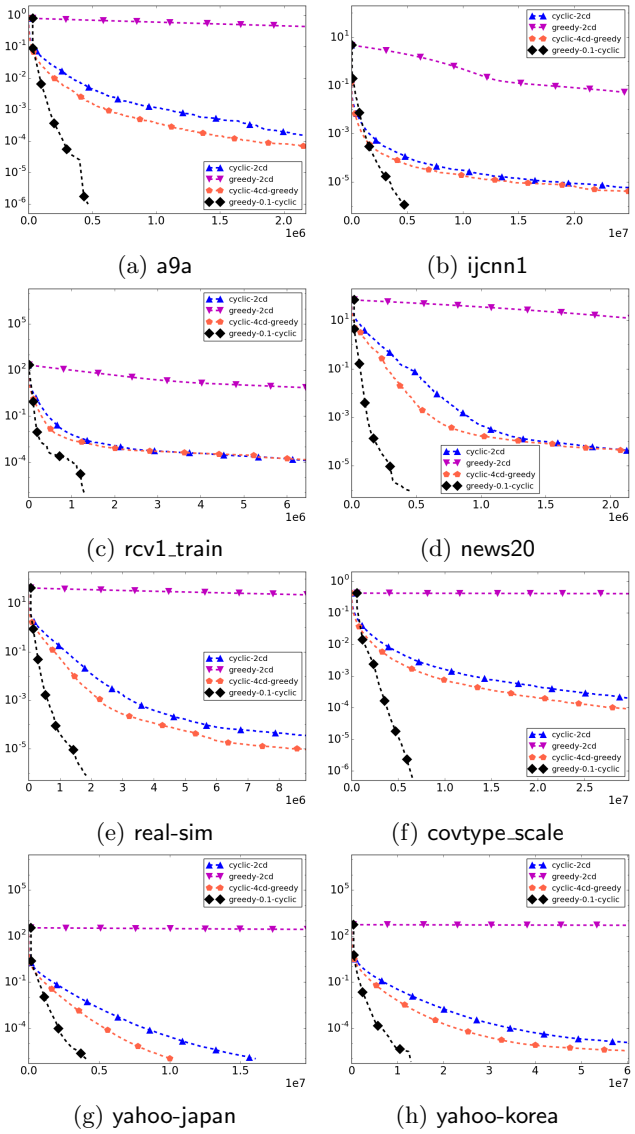
(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 5: 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).



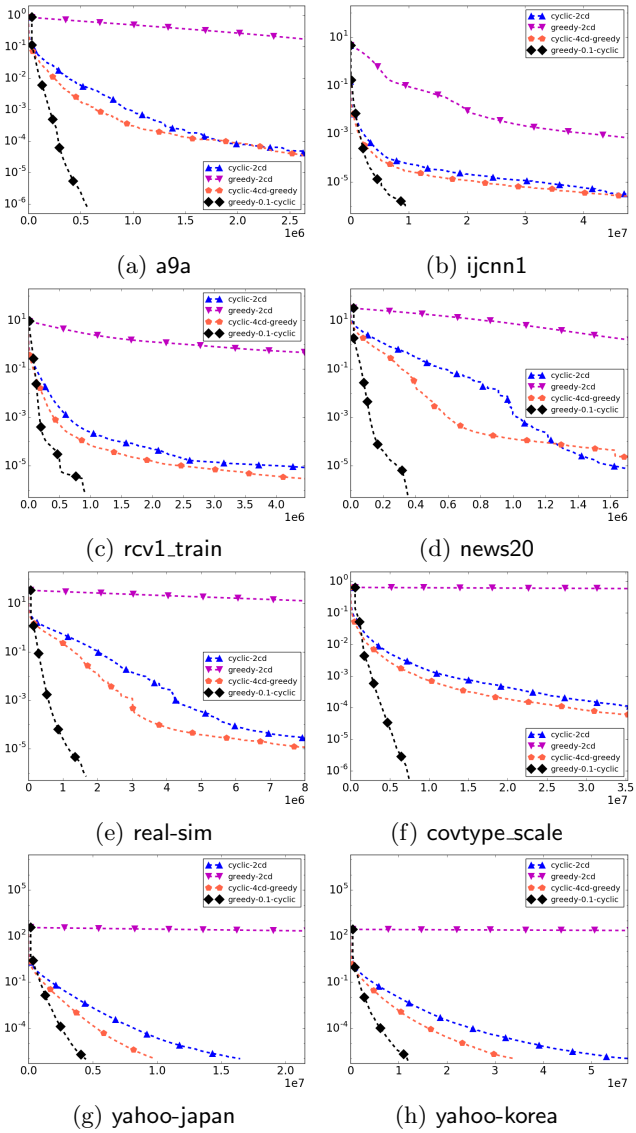
(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 6: A comparison on the convergence speed of different methods. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.01$ from Figure 5, 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).



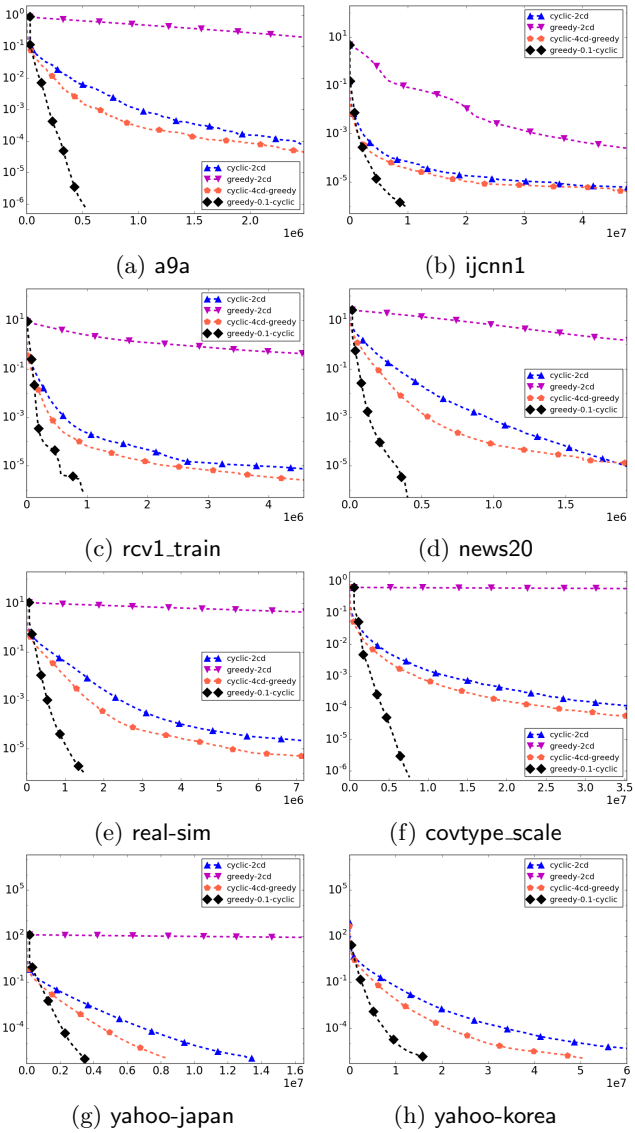
(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 7: 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).



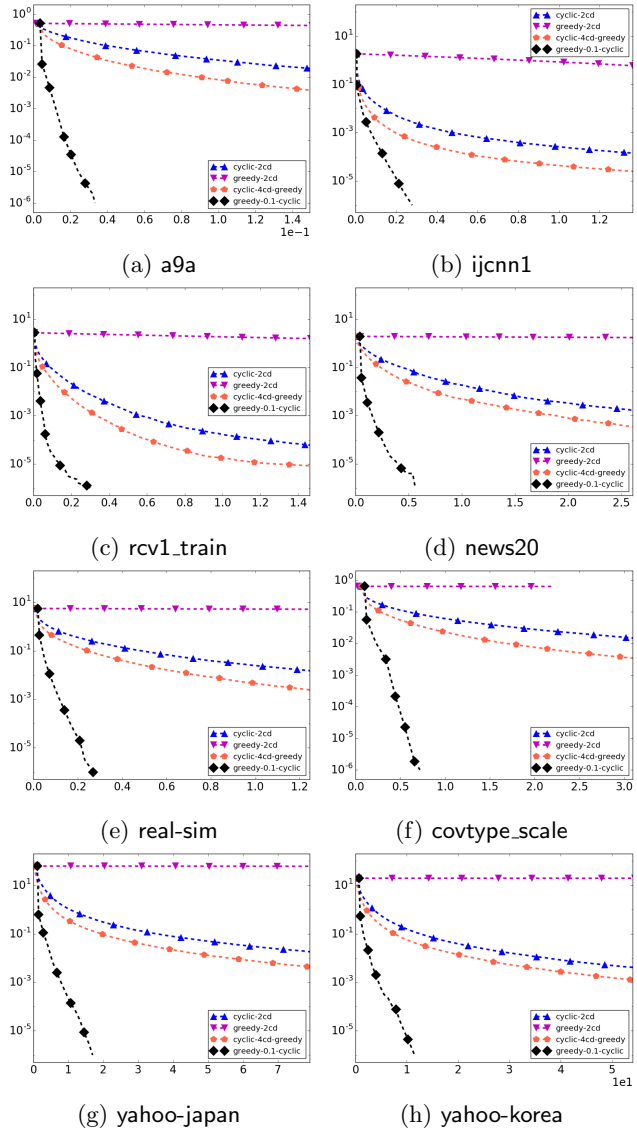
(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 8: A comparison on the convergence speed of different methods. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.005$ from Figure 7, 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).



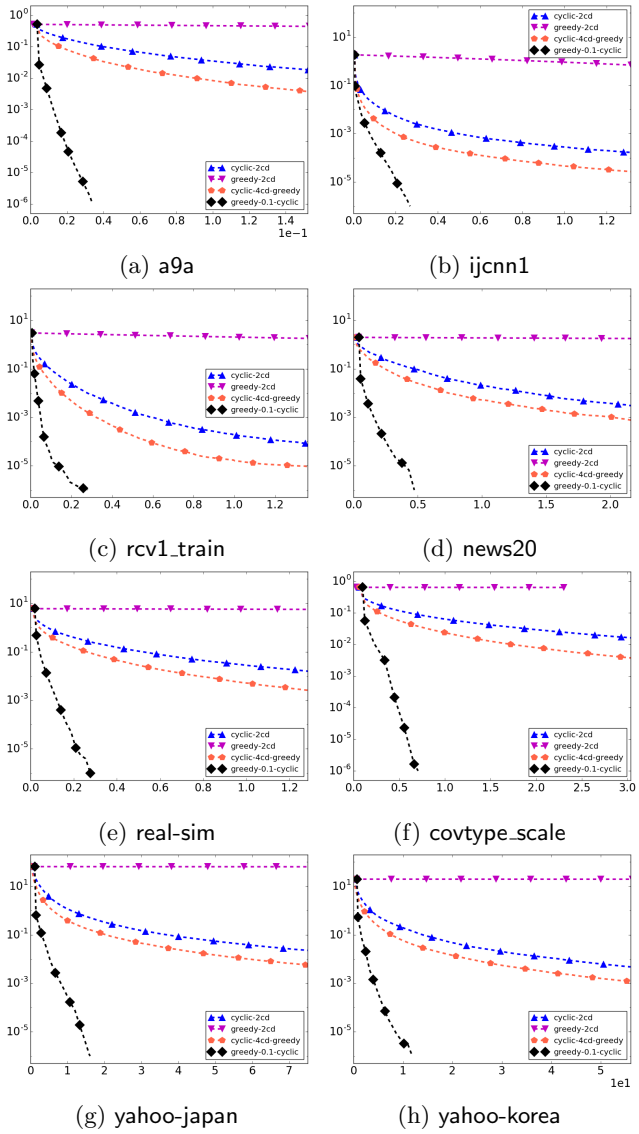
(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 9: 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).



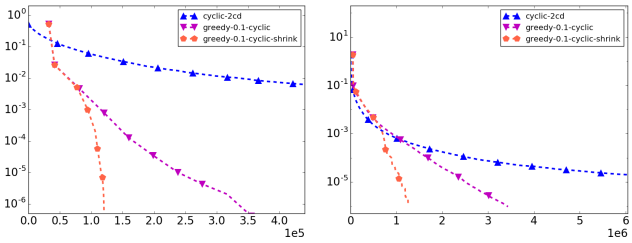
(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 10: A comparison on the convergence speed of different methods. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.1$ from Figure 9, 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).



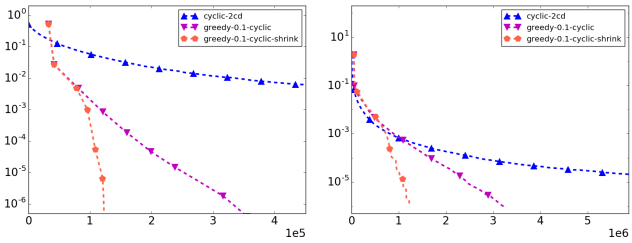
(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 11: 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).



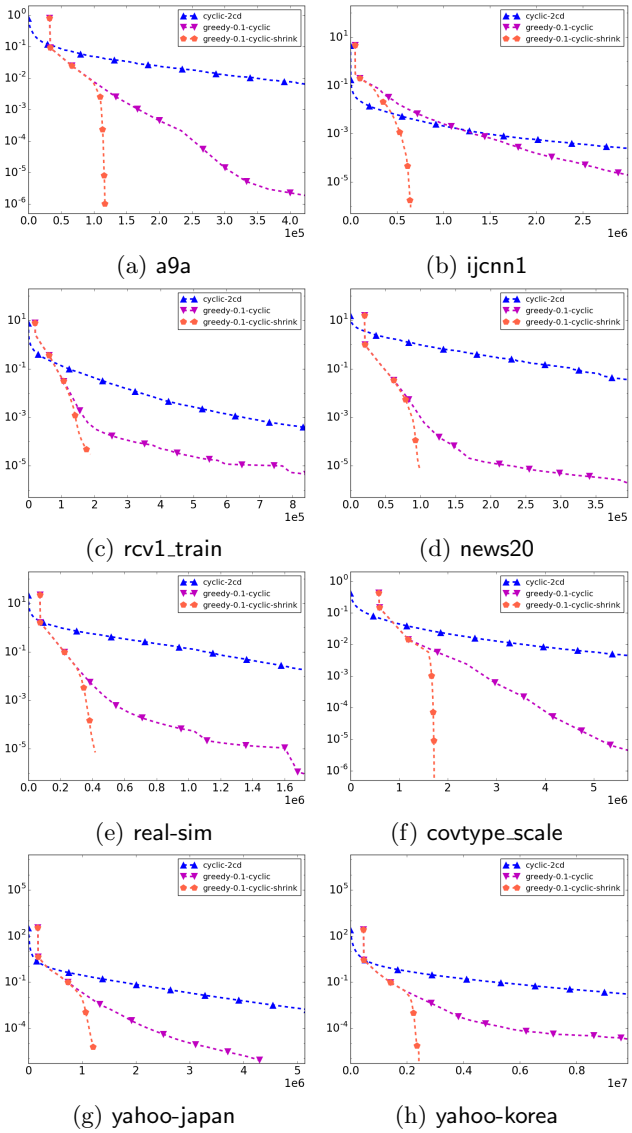
(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 12: 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 11, 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).



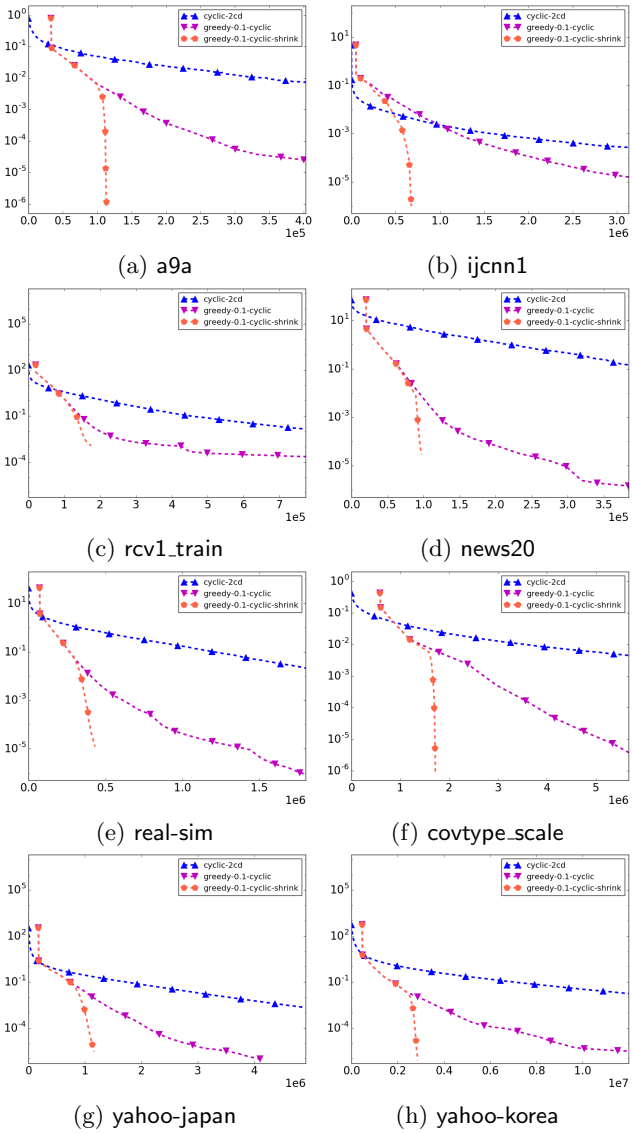
(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 13: 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).



(a) a9a (b) ijcnn1

(c) rcv1_train (d) news20

(e) real-sim (f) covtype_scale

(g) yahoo-japan (h) yahoo-korea

Figure 14: 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 13, 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).

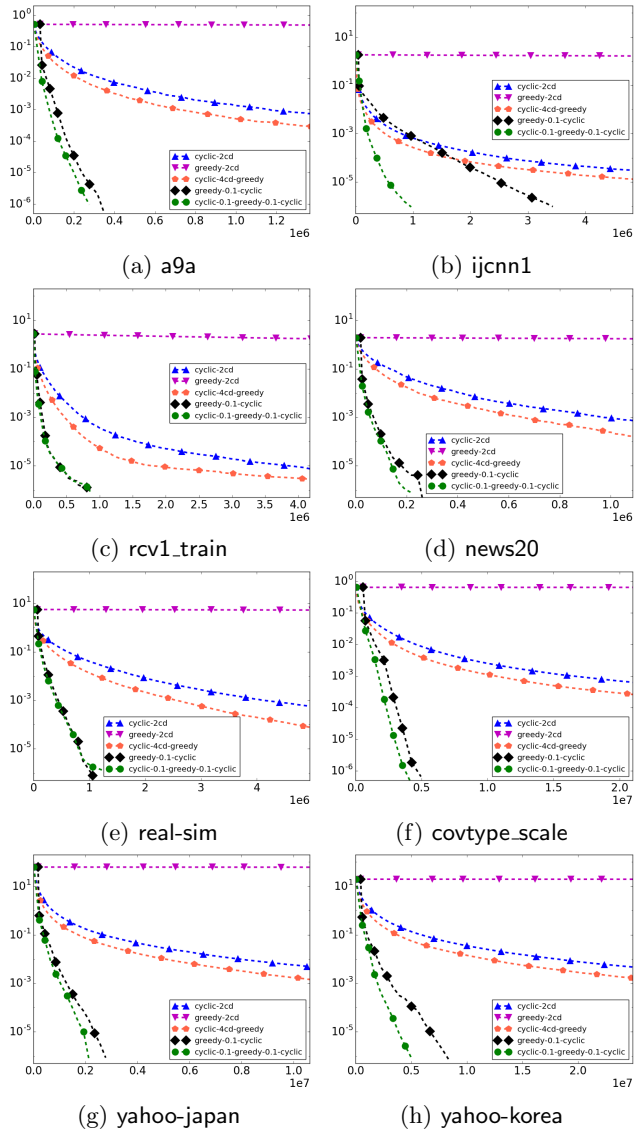
1: Let $\boldsymbol{\alpha}$ be a feasible point and calculate $Q_{ii}, \forall i$
2: **while** $\boldsymbol{\alpha}$ is not optimal **do**
3: Select $\bar{B}$
4: Calculate $\nabla_{\bar{B}} f(\boldsymbol{\alpha})$
5: Select the r most violating indices as B
6: $B = \{i_1, \ldots, i_r\}$
7: **for** $s = 1, \ldots, r$ **do**
8: Update α_{i_s}
9: **end for**
10: **end while**

(a) Algorithm 5 in [3].

1: Let $\boldsymbol{\alpha}$ be a feasible point and calculate $Q_{ii}, \forall i$
2: **while** $\boldsymbol{\alpha}$ is not optimal **do**
3: Select $\bar{B}$
4: Calculate $\nabla_{\bar{B}} f(\boldsymbol{\alpha})$
5: Select the r most violating pairs as B
6: $B = \{i_1, j_1\} \cup \cdots \cup \{i_r, j_r\}$
7: **for** $s = 1, \ldots, r$ **do**
8: Update $\alpha_{i_s}, \alpha_{j_s}$
9: **end for**
10: **end while**

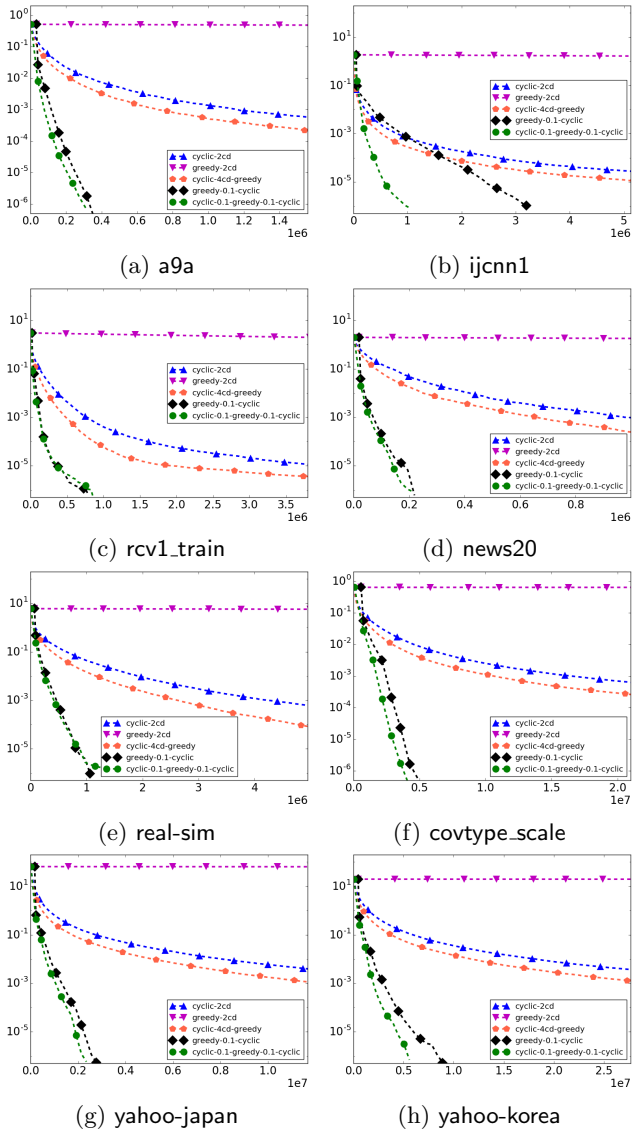
(b) A simplified description of Algorithm 8.

Figure 15: A line-by-line comparison between an algorithm in [3] and our Algorithm 8



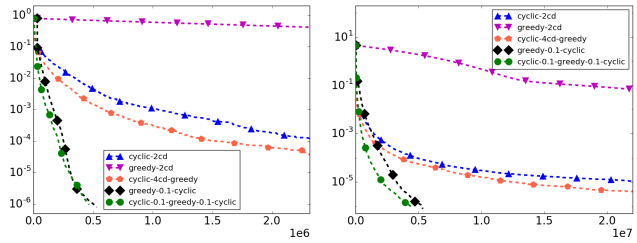
(a) a9a (b) ijcnn1 (c) rcv1_train (d) news20 (e) real-sim (f) covtype_scale (g) yahoo-japan (h) yahoo-korea

Figure 16: 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).

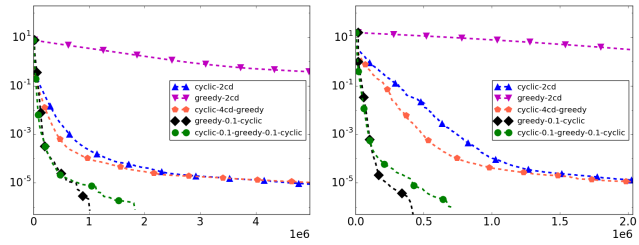


(a) a9a (b) ijcnn1 (c) rcv1_train (d) news20 (e) real-sim (f) covtype_scale (g) yahoo-japan (h) yahoo-korea

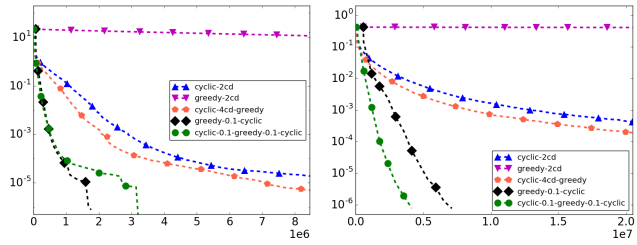
Figure 17: A comparison on the convergence speed of Algorithm 8 and other methods. Linear SVDD with $C = 1/(\nu l)$, where $\nu = 0.1$ from Figure 16, is used. The x-axis is the cumulative number of $O(n)$ operations, while the y-axis is the relative difference to the optimal function value defined in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



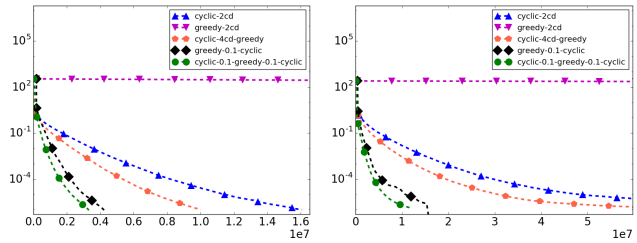
(a) a9a (b) ijcnn1



(c) rcv1_train (d) news20

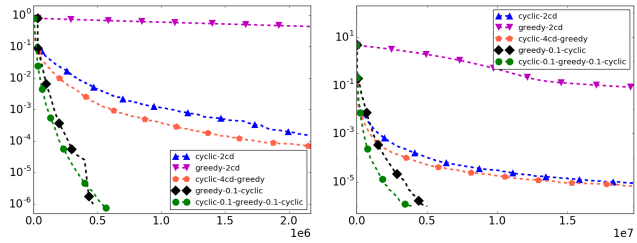


(e) real-sim (f) covtype_scale

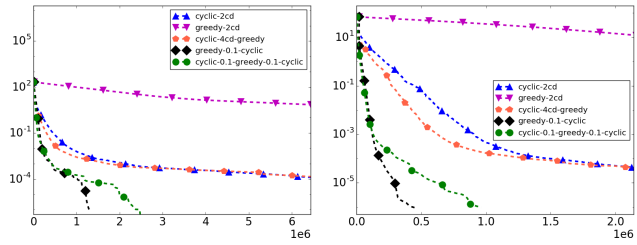


(g) yahoo-japan (h) yahoo-korea

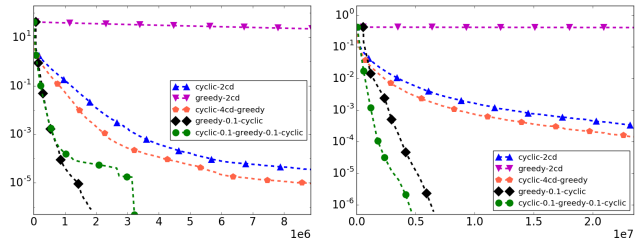
Figure 18: A comparison on the convergence speed of Algorithm 8 and other methods. Linear one-class SVM with $\nu = 0.01$ is considered. The x-axis is the cumulative number of $O(n)$ operations, while the y-axis is the relative difference to the optimal function value defined in (4.19).



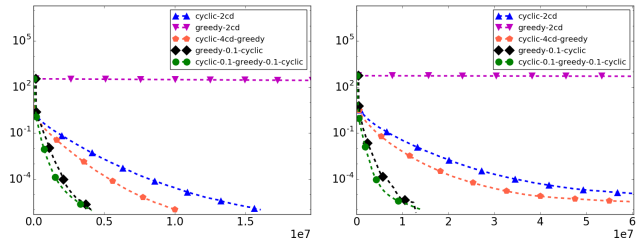
(a) a9a (b) ijcnn1



(c) rcv1_train (d) news20



(e) real-sim (f) covtype_scale



(g) yahoo-japan (h) yahoo-korea

Figure 19: 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 18, 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).