Supplement Materials for "Large-scale Linear RankSVM"

Ching-Pei Lee and Chih-Jen Lin

Department of Computer Science, National Taiwan University, Taipei 10617, Taiwan

I Introduction

This document presents some materials not included in the paper. In Section II, we illustrate the direct method for computing $l_i^+(\boldsymbol{w}), l_i^-(\boldsymbol{w}), \alpha_i^+(\boldsymbol{w}, \boldsymbol{v})$ and $\alpha_i^-(\boldsymbol{w})$, as well as the approach in Joachims (2006) that is similar to this method. Section III gives a comparison on relative function value, pairwise accuracy and NDCG with respect to the number of (CG) iterations between TRON and the cutting plane method used in TreeRankSVM. The results show that despite of their implementation differences, TRON has better convergence than the cutting plane method. In Section IV we discuss the possibility of solving the dual problem.

II Direct Methods to Calculate $l_i^+(\boldsymbol{w}), l_i^-(\boldsymbol{w}), \alpha_i^+(\boldsymbol{w}, \boldsymbol{v})$ and $\alpha_i^-(\boldsymbol{w}, \boldsymbol{v})$

In this section, we describe the direct method mentioned in Section 2.2 in detail, and then introduce a variant used in Joachims (2006).

II.1 A Direct Counting Method

To find $l_i^+(\boldsymbol{w})$, we must count the cardinality of the following set.

$$\mathrm{SV}_i^+(\boldsymbol{w}) = \{j \mid y_j > y_i, \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_i + 1\}.$$

The main difficulty is that both the order of y_i and the order of $\boldsymbol{w}^T\boldsymbol{x}_i$ are involved. We can first sort $\boldsymbol{w}^T\boldsymbol{x}_i$ in ascending order. For easier description, we assume that

$$\boldsymbol{w}^T\boldsymbol{x}_1 \leq \cdots \leq \boldsymbol{w}^T\boldsymbol{x}_l. \tag{II.1}$$

We then notice that if

$$\mathrm{count}^+(r) \equiv |\{j \mid y_j = r, \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_i + 1\}|, \ \forall r \in K$$

are available, then

$$l_i^+(\boldsymbol{w}) = \sum_{r:r>y_i} \text{count}^+(r). \tag{II.2}$$

From (5), we can easily maintain $\text{count}^+(r)\ \forall r$ when moving from i to $i+1$ by

$$\text{count}^+(y_j) \leftarrow \text{count}^+(y_j) + 1, \quad \text{if } \boldsymbol{w}^T\boldsymbol{x}_i + 1 \le \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_{i+1} + 1,$$

because

$$\{j \mid \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_i + 1\} \subset \{j \mid \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_{i+1} + 1\}.$$

We illustrate how this method obtains $l_i^+(\boldsymbol{w})$ by considering the following example.

i	1	2	3	4	5	6
$\boldsymbol{w}^T\boldsymbol{x}_i$	−0.9	−0.7	−0.1	0.15	0.2	1.6
y_i	2	1	2	3	3	2

For $i = 1$, we have

$$\begin{array}{cccccc} \downarrow & & & \downarrow & & \\ -0.9 & -0.7 & -0.1 & 0.15 & 0.2 & 1.6, \end{array}$$

where the first pointer indicates the current i, while the second one indicates the bound $\boldsymbol{w}^T\boldsymbol{x}_i + 1$. Thus, $\text{count}^+ = (1, 2, 0)$, and $l_1^+(\boldsymbol{w}) = 0$. For $i = 2$, we have

$$\begin{array}{cccccc} & \downarrow & & & \downarrow & \\ -0.9 & -0.7 & -0.1 & 0.15 & 0.2 & 1.6, \end{array}$$

and $\text{count}^+ = (1, 2, 2)$. Thus, $l_2^+(\boldsymbol{w}) = 2 + 2 = 4$.

The calculation for $l_i^-(\boldsymbol{w})$ is similar but goes through the whole data from l to 1 and maintains $|\{j \mid y_j = r, \boldsymbol{w}^T\boldsymbol{x}_j > \boldsymbol{w}^T\boldsymbol{x}_i - 1\}|,\ \forall r$.

Next, we discuss how to calculate $\alpha_i^+(\boldsymbol{w}, \boldsymbol{v})$ and $\alpha_i^-(\boldsymbol{w}, \boldsymbol{v})$. Notice that

$$l_i^+(\boldsymbol{w}) = \sum_{j \in \text{SV}_i^+(\boldsymbol{w})} 1, \quad \text{and} \quad \alpha_i^+(\boldsymbol{w}, \boldsymbol{v}) = \sum_{j \in \text{SV}_i^+(\boldsymbol{w})} \boldsymbol{x}_i^T\boldsymbol{v}. \tag{II.3}$$

Thus, $\alpha_i^+(\boldsymbol{w}, \boldsymbol{v})$ can be calculated by a similar method that maintains the following value.

$$\text{xv}^+(r) \equiv \sum_{j:y_j=r, \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_i+1} \boldsymbol{x}_j^T\boldsymbol{v},\ \forall r \in K.$$

If $\boldsymbol{w}^T\boldsymbol{x}_i$ have been sorted before CG iterations, this approach needs $O(l+k)$ space and costs

$$O(l\bar{n} + lk + n) \tag{II.4}$$

time for one matrix-vector product. This type of approach has been used by Joachims (2005) and Chapelle and Keerthi (2010) for the situation of $k = 2$, although our discussion is more general for any k. A procedure with similar complexity for general k has been proposed by Joachims (2006). See more discussion in Section II.2.

Because $O(lk) \le O(p) = O(l^2)$, the current approach is better than the method by (12). However, the $O(lk)$ complexity is still high if k is large.

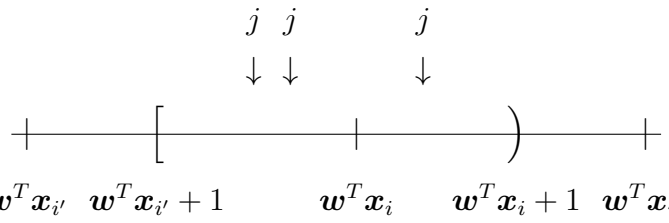


Figure (I): An illustration of the counting approach of Joachims (2006)

II.2 Details of the Method in Joachims (2006)

Joachims (2006) considered a direct method that is similar to that in Section II.1 to compute $l_i^+(\boldsymbol{w})$ and $l_i^-(\boldsymbol{w})$. We assume the data has been sorted as in (II.1). Each time this method considers a relevance level r and calculates

$$l_i^+(\boldsymbol{w}),\ \forall i \text{ with } y_i = r.$$

Assume $r = r_0$ and

$$\boldsymbol{w}^T\boldsymbol{x}_{i'} \le \boldsymbol{w}^T\boldsymbol{x}_i \le \boldsymbol{w}^T\boldsymbol{x}_{i''} \text{ with } y_{i'} = y_i = y_{i''} = r_0$$

are three consecutive elements with relevance level r_0. We have the situation in Figure (I), and

$$\begin{aligned}
&l_i^+(\boldsymbol{w}) \\
&= |\{j \mid y_j > r_0,\ \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_i + 1\}| \\
&= |\{j \mid y_j > r_0,\ \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_{i'} + 1\}| + |\{j \mid y_j > r_0,\ \boldsymbol{w}^T\boldsymbol{x}_{i'} + 1\ \le \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_i + 1\}| \\
&= l_{i'}^+(\boldsymbol{w}) + |\{j \mid y_j > r_0,\ \boldsymbol{w}^T\boldsymbol{x}_{i'} + 1 \le \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_i + 1\}|.
\end{aligned}$$

Therefore, if $l_{i'}^+(\boldsymbol{w})$ is known, one only needs to count those indices j satisfying

$$y_j > r_0 \quad \text{and} \quad \boldsymbol{w}^T\boldsymbol{x}_{i'} + 1 \le \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_i + 1.$$

They are elements in the semi-open interval in Figure (I). Joachims (2006) carefully observes that simultaneously one can also count elements in $l_j^-(\boldsymbol{w})$ by the following formulation.

$$\begin{aligned}
l_j^-(\boldsymbol{w}) &= |\{s \mid y_j > y_s,\ \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_s + 1\}| \\
&= \sum_{r:r<y_j} |\{s \mid y_s = r,\ \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_s + 1\}|. \qquad \text{(II.5)}
\end{aligned}$$

From Figure (I), because j is in the semi-open interval, the smallest $\boldsymbol{w}^T\boldsymbol{x}_s$ satisfying $\boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_s + 1$ and $y_s = r_0$ is $\boldsymbol{w}^T\boldsymbol{x}_i$. Therefore,

$$\{s \mid y_s = r,\ \boldsymbol{w}^T\boldsymbol{x}_j < \boldsymbol{w}^T\boldsymbol{x}_s + 1\} = \{s \mid y_s = r,\ \boldsymbol{w}^T\boldsymbol{x}_s \ge \boldsymbol{w}^T\boldsymbol{x}_i\}. \qquad \text{(II.6)}$$

If the cardinality of the set (II.6)

$$|\{s \mid y_s = r,\ \boldsymbol{w}^T\boldsymbol{x}_s \ge \boldsymbol{w}^T\boldsymbol{x}_i\}| \qquad \text{(II.7)}$$

Algorithm (I) The method in Joachims (2006) for calculating $l_i^+(\boldsymbol{w})$ and $l_i^-(\boldsymbol{w})$

1. Assume $K = \{1, \ldots, k\}$. $l_i^-(\boldsymbol{w}) \leftarrow 0, \forall i = 1, \ldots l$.
2. Given X and $\boldsymbol{w}$, compute $X\boldsymbol{w}$.
3. Sort $\boldsymbol{w}^T\boldsymbol{x}_i$ in ascending order: $\boldsymbol{w}^T\boldsymbol{x}_{\pi(1)} \le \cdots \le \boldsymbol{w}^T\boldsymbol{x}_{\pi(l)}$.
4. For $r = 2, \ldots, k$
   - 4.1. $j \leftarrow 1$, $\text{count}^+ \leftarrow 0$, $\text{count}^- \leftarrow |\{s \mid y_s = r\}|$.
   - 4.2. For $i = 1, \ldots, l$
     - 4.2.1. If $y_{\pi(i)} = r$
       - a. While $j \le l$ and $1 - \boldsymbol{w}^T\boldsymbol{x}_{\pi(j)} + \boldsymbol{w}^T\boldsymbol{x}_{\pi(i)} > 0$
         - If $y_{\pi(j)} > r$
           - $\text{count}^+ \leftarrow \text{count}^+ + 1$.
           - $l^-_{\pi(j)} \leftarrow l^-_{\pi(j)} + \text{count}^-$.
         - $j \leftarrow j + 1$.
       - b. $l_i^+(\boldsymbol{w}) \leftarrow \text{count}^+$.
       - c. $\text{count}^- \leftarrow \text{count}^- - 1$.

can be maintained, it can be used for calculating $l_j^-(\boldsymbol{w})$ in (II.5). To update (II.7), if i is moved to i'', clearly

$$|\{s \mid y_s = r_0,\ \boldsymbol{w}^T\boldsymbol{x}_s \ge \boldsymbol{w}^T\boldsymbol{x}_{i''}\}| = |\{s \mid y_s = r_0,\ \boldsymbol{w}^T\boldsymbol{x}_s \ge \boldsymbol{w}^T\boldsymbol{x}_i\}| - 1.$$

The initial value of (II.7) is

$$|\{s \mid y_s = r\}|.$$

In summary, the method by Joachims (2006) goes through all $r \in S$ and conducts operations described above for i from 1 to l. Thus the cost is $O(lk)$ for computing $l_i^+(\boldsymbol{w})$ and $l_i^-(\boldsymbol{w}), \forall i$. The detailed procedure was in Algorithm 2 of Joachims (2006), but we rewrite it in Algorithm (I) by our notation.

This procedure is related to that in Section II.1. They differ in several aspects. The algorithm here goes through all data k times, while the procedure in Section II.1 needs only two passes, one for $l_i^+(\boldsymbol{w}),\ \forall i$ and another for $l_i^-(\boldsymbol{w}),\ \forall i$. However, the space needed here is only $O(l)$, which is less than $O(k + l)$ in Section II.1 for maintaining $\text{count}^+(r),\ \forall r \in S$.

III A Comparison Between TreeRankSVM and Tree-TRON in Number of Iterations

In Section 4.4, we observed that the cutting plane method in TreeRankSVM is slower than TRON. Because implementation details may affect the running time, we examine the relative function values, pairwise accuracy and NDCG versus number of iterations in Figure (II) by comparing Tree-TRON and TreeRankSVM. For Tree-TRON, we use CG iterations rather than outer Newton iterations because each CG has a similar complexity to that of an iteration in the cutting plane method. The parameters used are the same as those for best pairwise accuracy

in Table 3. Because TreeRankSVM solves (2) while Tree-TRON solves (3), their selected regularization parameters may be different. Similar to the experiments in Section 4.4, we draw a horizontal line to indicate the performance of Tree-TRON using its default stopping condition. In all problems Tree-TRON is faster, so this experiment indicates that TRON is more suitable than cutting plane methods for linear rankSVM.

IV The Possibility of Solving the Dual Problem

When nonlinear mappings are considered, methods for optimizing the standard SVMs usually utilize the kernel function to solve the dual problem. From Appendix A, the dual problems of (2) and (3) both have p variables. It is difficult to solve such problems if $p = O(l^2)$. However, by removing some zero variables during the optimization procedure, we may efficiently solve a smaller problem. If L2 loss is used, from Appendix A, we have

$$\boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j) \geq 1 \quad \Leftrightarrow \quad \text{the corresponding dual variable is zero.}$$

If at the optimal solution, two pairs (i, j) and (j, s) in P are in the correct order with

$$\boldsymbol{w}^T\boldsymbol{x}_i - \boldsymbol{w}^T\boldsymbol{x}_j \geq 1 \quad \text{and} \quad \boldsymbol{w}^T\boldsymbol{x}_j - \boldsymbol{w}^T\boldsymbol{x}_s \geq 1,$$

then

$$\boldsymbol{w}^T\boldsymbol{x}_i - \boldsymbol{w}^T\boldsymbol{x}_s \geq 1 \tag{IV.1}$$

and the corresponding dual variable is zero. Therefore, in the best situation, only $O(l)$ of the p variables are zero. Then the situation is similar to that of using partial pairs in Section 5. To check how good the sparsity is in practice, in Table (I) we present the percentage of zero elements of the dual optimal solution. We use a large penalty parameter $C = 2^{20}$ to get small training errors and therefore, better sparsity.[1] Unfortunately, the solution is very dense; more than 75% of the elements are non-zero. A further check shows that while most pairs are correctly classified, they satisfy

$$0 < \boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j) < 1$$

only, so the corresponding dual variable is still non-zero. Therefore, solving the dual problem may not be an efficient option.

References

Olivier Chapelle and S. Sathiya Keerthi. Efficient algorithms for ranking with SVMs. *Information Retrieval*, 13(3):201–215, 2010.

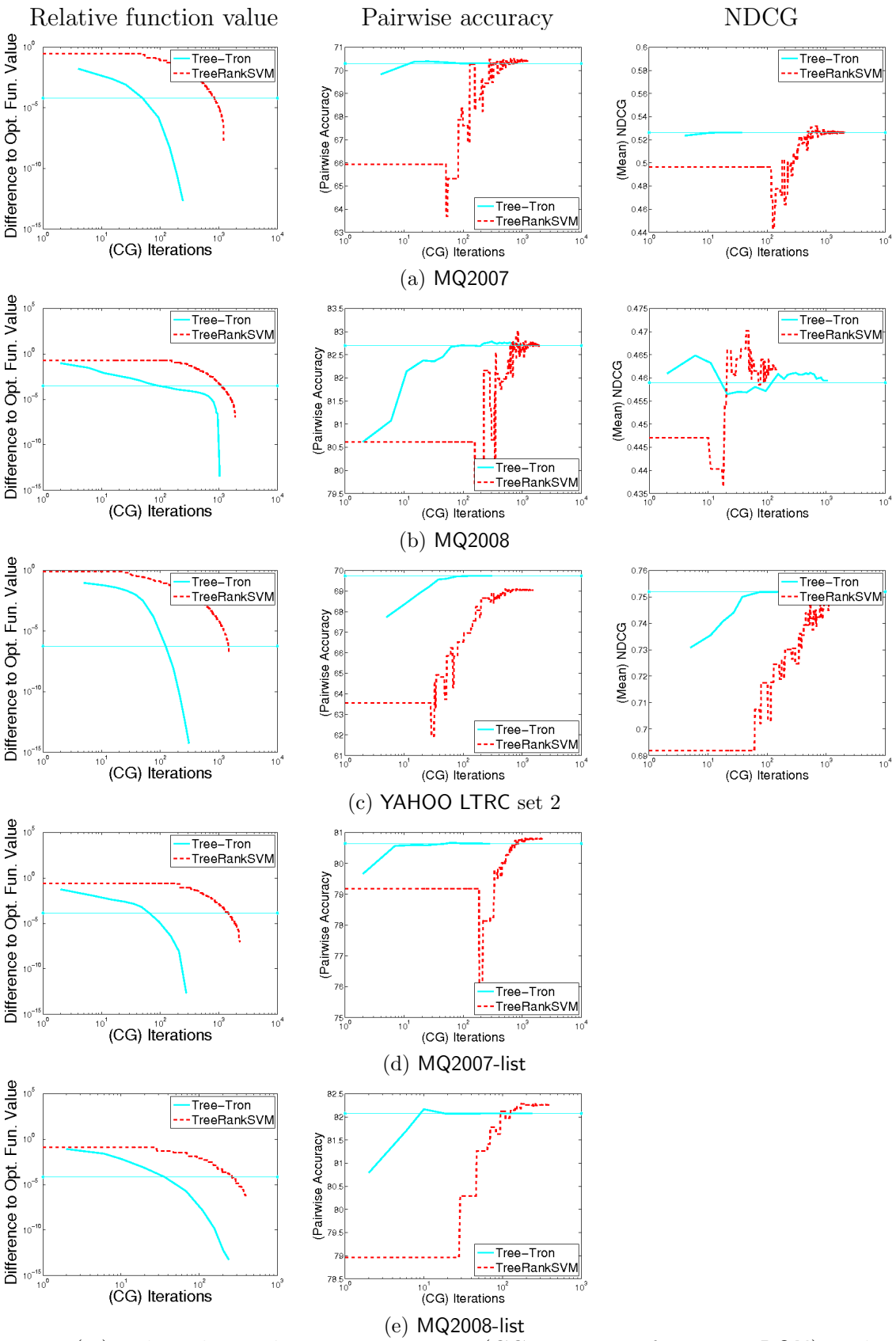
[1]Because of the lengthy training time of using parameter $C = 2^{20}$, we report results of only five data sets.

Data sets	Percentage of zero dual variables
MQ2007	5.15%
MQ2008	22.44%
YAHOO LTRC set 2	11.14%
MQ2007-list	19.64%
MQ2008-list	23.86%

Table (I): Sparsity of the dual optimal solution of L2-loss linear rankSVM. $C = 2^{20}$ is used.

Thorsten Joachims. A support vector method for multivariate performance measures. In *Proceedings of the Twenty Second International Conference on Machine Learning (ICML)*, 2005.

Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.



(a) MQ2007

(b) MQ2008

(c) YAHOO LTRC set 2

(d) MQ2007-list

(e) MQ2008-list

Figure (II): The relation between iterations (CG iterations for Tree-TRON) and function values, pairwise accuracy and NDCG.