HPC Project: Distributed Linear rankSVM

Ya Zhu

May 17, 2017

1 Problem and major computation

RankSVM, an extension of standard support vector machines, is widely used to solve "learning to rank" problem in information retrieval, recommender systems, online advertising, etc.. When dealing with large-scale data, the training time of rankSVM is lengthy. So developing a parallel or distributed algorithm is necessary.

Given Training instances $\{(y_i, q_i, \boldsymbol{x}_i)\}_{i=1}^{l}$, $y_i \in K \subset \mathbf{R}$, $|K| = k$, $q_i \in \{1, \ldots, q\}$, $\boldsymbol{x}_i \in \mathbf{R}^n$ where $y_i > y_j$ means i is more preferred than j, and q_i is the ith query represented by an integer here, our goal is to find a vector $\boldsymbol{w}$ such that $\boldsymbol{w}^T\boldsymbol{x}_i > \boldsymbol{w}^T\boldsymbol{x}_j, \forall y_i > y_j, q_i = q_j$. L2-loss linear rankSVM finds such a $\boldsymbol{w}$ by solving the following problem:

$$\min_{\boldsymbol{w}} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{(i,j)\in P} \max\left(0, 1 - \boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j)\right)^2, \tag{1}$$

where P is defined as Preference pairs: $P \equiv \{(i,j) | q_i = q_j, y_i > y_j\}$.

There are many optimization methods for solving (1). For instance, [Lee and Lin, 2014] used a trust region Newton method. Assume $f(\boldsymbol{w})$ is twice-differentiable, a trust region Newton method iteratively minimizes its 2nd-order apporximation

$$\min_{\boldsymbol{w}} \quad \nabla f(\boldsymbol{w}^t)^T\boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T\nabla^2 f(\boldsymbol{w}^t)\boldsymbol{s}, \|\boldsymbol{s}\| \le \Delta_t$$

with a given Δ_t in the t-th iteration. Then solve the equivalent linear system

$$\nabla^2 f(\boldsymbol{w}^t)\boldsymbol{s} = -\nabla f(\boldsymbol{w}^t)$$

by conjugate gradient method to obtain $\boldsymbol{s}^t$ and update $\boldsymbol{w}^{t+1}$ correspondingly using trust region method.

Conjugate gradient (CG) is an iterative method, at each CG iteration, it involves the computation of

$$f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\boldsymbol{w}^T X(A_{\boldsymbol{w}}^T A_{\boldsymbol{w}} X\boldsymbol{w} - 2A_{\boldsymbol{w}}^T\boldsymbol{e}_{\boldsymbol{w}} + p_{\boldsymbol{w}}), \tag{2}$$

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + 2C\boldsymbol{w}^T X^T(A_{\boldsymbol{w}}^T A_{\boldsymbol{w}} X\boldsymbol{w} - A_{\boldsymbol{w}}^T\boldsymbol{e}_{\boldsymbol{w}}), \tag{3}$$

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2CX^T A_{\boldsymbol{w}}^T A_{\boldsymbol{w}} X\boldsymbol{v}, \tag{4}$$

where

$$A_{\boldsymbol{w}} \equiv \begin{array}{c c} & \begin{array}{ccccc} \cdots & i & \cdots & j & \cdots \end{array} \\ \begin{array}{c} \vdots \\ (i,j) \\ \vdots \end{array} & \left[\begin{array}{ccccc} & & & & \\ 0\cdots0 & +1 & 0\cdots0 & -1 & 0\cdots0 \\ & & & & \end{array}\right] \end{array},$$

with all $(i,j) \in \mathrm{SV}(\boldsymbol{w})$. $\mathrm{SV}(\boldsymbol{w})$ is defined as

$$\mathrm{SV}(\boldsymbol{w}) \equiv \{(i,j) \mid (i,j) \in P, 1 - \boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j) > 0\}.$$

If define

$$\begin{aligned} l_i^+(\boldsymbol{w}) &\equiv |\{j \mid (j,i) \in \mathrm{SV}(\boldsymbol{w})\}|, \\ l_i^-(\boldsymbol{w}) &\equiv |\{j \mid (i,j) \in \mathrm{SV}(\boldsymbol{w})\}|, \end{aligned}$$

then the common terms in (2)-(4) can be written as follows:

$$A_{\boldsymbol{w}}^T \boldsymbol{e}_{\boldsymbol{w}} = \begin{bmatrix} (l_1^- - l_1^+) \\ \vdots \\ (l_l^- - l_l^+) \end{bmatrix}, \tag{5}$$

$$p_{\boldsymbol{w}} = \sum_{i=1}^{l} l_i^+ = \sum_{i=1}^{l} l_i^-, \tag{6}$$

$$A_{\boldsymbol{w}}^T A_{\boldsymbol{w}} X \boldsymbol{w} = \begin{bmatrix} (l_1^+ + l_1^-)\boldsymbol{w}^T\boldsymbol{x}_1 - (\beta_1^+ + \beta_1^-) \\ \vdots \\ (l_l^+ + l_l^-)\boldsymbol{w}^T\boldsymbol{x}_l - (\beta_l^+ + \beta_l^-) \end{bmatrix}. \tag{7}$$

$$X^T A_{\boldsymbol{w}}^T A_{\boldsymbol{w}} X \boldsymbol{v} = X^T \begin{bmatrix} (l_1^+ + l_1^-)\boldsymbol{x}_1^T\boldsymbol{v} - (\alpha_1^+ + \alpha_1^-) \\ \vdots \\ (l_l^+ + l_l^-)\boldsymbol{x}_l^T\boldsymbol{v} - (\alpha_l^+ + \alpha_l^-) \end{bmatrix}, \tag{8}$$

where l_i^+,l_i^-,α_i^+,α_i^-,l_β^+ and β_i^- can be computed by order-statistic trees in $O(l \log l)$ time [Airola et al., 2011, Lee and Lin, 2014].

2 Parallelization

The major task for parallelization here is to compute (5)-(8) in parallel with distributed stored instance matrix X. Actually the matrix X can be distributed stored in instance-wise (row-wise) or feature-wise (column-wise). For this problem, we can tell that instances with different queries can be treated independently, so we can partition the instance vectors in query-wise.

2.1 Query-wise partition

If we group the instance by query, and partition X uniformly so that all the instances with the same query are in the same partition, then for any mth machine, where $m = 1 \dots M$ is the index of machines (suppose there are M machines in total), we have

$$A_{m,\boldsymbol{w}}^T \boldsymbol{e}_{m,\boldsymbol{w}} = \begin{bmatrix} (l_1^- - l_1^+) \\ \vdots \\ (l_{l_m}^- - l_{l_m}^+) \end{bmatrix}, \quad p_{m,\boldsymbol{w}} = \sum_{i=1}^{l_m} l_i^+ = \sum_{i=1}^{l_m} l_i^-,$$

$$A_{m,\boldsymbol{w}}^T A_{m,\boldsymbol{w}} X_m \boldsymbol{w} = \begin{bmatrix} (l_1^+ + l_1^-)\boldsymbol{w}^T\boldsymbol{x}_1 - (\beta_1^+ + \beta_1^-) \\ \vdots \\ (l_{l_m}^+ + l_{l_m}^-)\boldsymbol{w}^T\boldsymbol{x}_{l_m} - (\beta_{l_m}^+ + \beta_{l_m}^-) \end{bmatrix}.$$

$$X_m^T A_{m,\boldsymbol{w}}^T A_{m,\boldsymbol{w}} X_m \boldsymbol{v} = X_m^T \begin{bmatrix} (l_1^+ + l_1^-)\boldsymbol{x}_1^T \boldsymbol{v} - (\alpha_1^+ + \alpha_1^-) \\ \vdots \\ (l_{l_m}^+ + l_{l_m}^-)\boldsymbol{x}_{l_m}^T \boldsymbol{v} - (\alpha_{l_m}^+ + \alpha_{l_m}^-) \end{bmatrix},$$

and in total, we have

$$A_{m,\boldsymbol{w}}^T \boldsymbol{e}_{m,\boldsymbol{w}} = \begin{bmatrix} A_{1,\boldsymbol{w}} \\ \vdots \\ A_{M,\boldsymbol{w}} \end{bmatrix}, \quad p_{\boldsymbol{w}} = \sum_{i=m}^{M} p_{m,\boldsymbol{w}}$$

$$A_{\boldsymbol{w}}^T A_{\boldsymbol{w}} X \boldsymbol{w} = \begin{bmatrix} A_{1,\boldsymbol{w}}^T A_{1,\boldsymbol{w}} X_1 \boldsymbol{w} \\ \vdots \\ A_{M,\boldsymbol{w}}^T A_{M,\boldsymbol{w}} X_M \boldsymbol{w} \end{bmatrix}.$$

$$X^T A_{\boldsymbol{w}}^T A_{\boldsymbol{w}} X \boldsymbol{v} = \begin{bmatrix} X_1^T A_{1,\boldsymbol{w}}^T A_{1,\boldsymbol{w}} X_1 \boldsymbol{v} \\ \vdots \\ X_M^T A_{M,\boldsymbol{w}}^T A_{M,\boldsymbol{w}} X_M \boldsymbol{v} \end{bmatrix}.$$

Thus, we can compute (2)-(4) by

$$f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\bigoplus_{m=1}^{M}(\boldsymbol{w}^T X_m^T \quad (A_{m,\boldsymbol{w}}^T A_{(m,\boldsymbol{w}} X_m \boldsymbol{w} - 2A_{m,\boldsymbol{w}}^T \boldsymbol{e}_{m,\boldsymbol{w}}) + p_{m,\boldsymbol{w}}), \tag{9}$$

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + 2C\bigoplus_{m=1}^{M} X_m^T (A_{m,\boldsymbol{w}}^T A_{m,\boldsymbol{w}} X_m \boldsymbol{w} - A_{m,\boldsymbol{w}}^T \boldsymbol{e}_{m,\boldsymbol{w}}), \tag{10}$$

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2C\bigoplus_{m=1}^{M} X_m^T A_{m,\boldsymbol{w}}^T A_{m,\boldsymbol{w}} X_{m,} \boldsymbol{v}. \tag{11}$$

where $\bigoplus$ is used to represent an *MPI_Allreduce* operation which collects computed values/vectors from all machines and redistributes the sum to them. We can see that for query-wise partition, the communication cost for one *MPI_Allreduce* is $O(n)$ (n is the number of dimensions of X).

2.2 Feature-wise partition

We can also partition X in feature-wise. That is, each machine store a part of instance vector $\boldsymbol{x}_i$ for all i. Thus each machine computes a partial result of $X\boldsymbol{w}$ and $X\boldsymbol{v}$, and then sum them up. Thus (2)-(4) would be computed as

$$f(\boldsymbol{w}) = \frac{1}{2}\bigoplus_{j=1}^{M}\boldsymbol{w}_j^T\boldsymbol{w}_j + C((\bigoplus_{j=1}^{M}\boldsymbol{w}_j^T X_j^T)(A_{\boldsymbol{w}}^T A_{\boldsymbol{w}}\bigoplus_{j=1}^{M} X_j\boldsymbol{w}_j - 2A_{\boldsymbol{w}}^T\boldsymbol{e}_{\boldsymbol{w}}) + p_{\boldsymbol{w}}), \tag{12}$$

and for the kth machine:

$$\nabla f(\boldsymbol{w})_k = \boldsymbol{w}_k + 2CX_k^T(A_{\boldsymbol{w}}^T A_{\boldsymbol{w}}\bigoplus_{j=1}^{M} X_j\boldsymbol{w}_j - A_{\boldsymbol{w}}^T\boldsymbol{e}_{\boldsymbol{w}}), \tag{13}$$

$$(\nabla^2 f(\boldsymbol{w})\boldsymbol{v})_k = \boldsymbol{v}_k + 2CX_k^T A_{\boldsymbol{w}}^T A_{\boldsymbol{w}}\bigoplus_{j=1}^{M} X_j\boldsymbol{v}_j. \tag{14}$$

So we can see that feature-wise partition is more complicated than query-time partition for it involves more communications, and note that l_i^+,l_i^-,α_i^+,α_i^-,l_β^+ and β_i^- cannot be calculated locally without communication.

Table 1: Data sets

Data set	l	n	$\|K\|$	q
`MQ2007`	42,158	46	3	1,017
`MQ2008`	9,630	46	3	471
`MQ2007-list`	743,790	46	1,268	1,017
`MQ2008-list`	540,679	46	1,831	471

Table 2: Communication cost of query-wise partition and feature-wise partition. The percentage shows the proportion of average communication time to total running time.

Data set	Query-wise	Feature-wise
`MQ2007`	62.71%	71.05%
`MQ2008`	58.93%	73.02%
`MQ2007-list`	6.39%	70.15%
`MQ2008-list`	10.09%	69.23%

3 Experiment

I modified the code from the RankSVM extension of LIBLINEAR[*] and implemented both query-wise and feature-wise parallel with MPI. The experiment is run on server `crunchy1` with compiler `openmpi-x86_64`.

The statistics of data sets used for experiments are illustrated in Table 1. They are all dense data sets with $l \gg n$. So query-wise partition may be more suitable for them.

Table 2 shows a comparison of average communication cost of query-wise partition and feature-wise partition. We can see that query-wise partition requires less communication cost than feature-wise one does. It also indicates that, for small data sets, the running time is partially bounded by communication while that of large data sets is not.

Figure 1 shows the scaling performance and speedup of the parallel algorithm with query-wise partition.

4 Conclusion

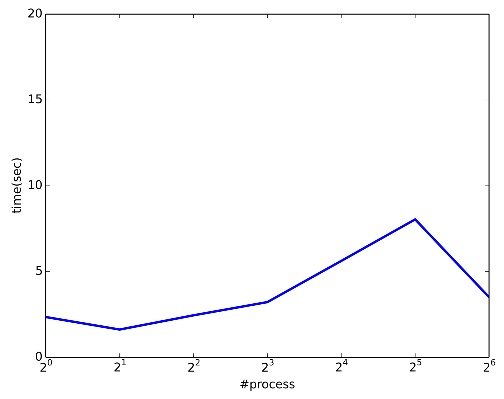
From previous analysis and the experiment results, we can conclude that

- Query-wise partition is more suitable when $l \gg n$ ($q \geq M$), and feature-wise is more suitable when $n \gg l$. For $M \gg q$, the communication cost of QW increases while FW does not.
- For large and dense data, the running time of the algorithm is dominated by computation cost and memory access, while for smaller and sparse data, it is dominated by communication cost.

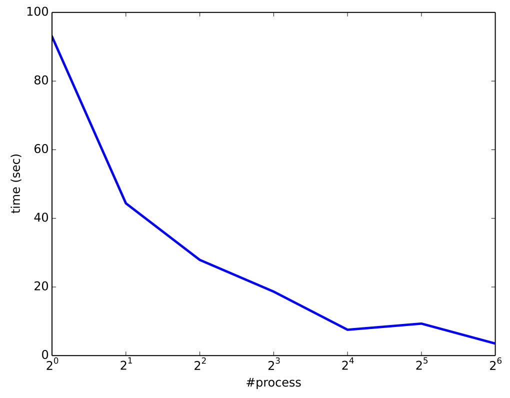
5 Future work

There are something I would like to do in the future such as: doing more experiments on both large dense and sparse data; optimize the memory access; extend to kernel rankSVM [Kuo et al., 2014].

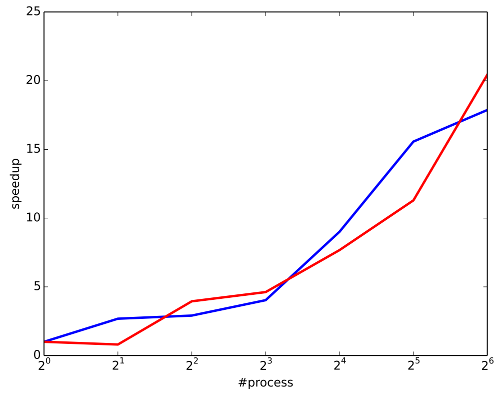
[*] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#large_scale_ranksvm`



(a) weak scaling



(b) strong scaling



(c) speedup with different algorithm parameters

Figure 1: Scaling and speedup of parallel linear rankSVM with query-wise partition.

References

A. Airola, T. Pahikkala, and T. Salakoski. Training linear ranking SVMs in linearithmic time using red–black trees. *Pattern Recognition Letters*, 32(9):1328–1336, 2011.

T.-M. Kuo, C.-P. Lee, and C.-J. Lin. Large-scale kernel rankSVM. In *Proceedings of SIAM International Conference on Data Mining*, 2014. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/ranksvm/kernel.pdf`.

C.-P. Lee and C.-J. Lin. Large-scale linear rankSVM. *Neural Computation*, 26(4):781–817, 2014. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/ranksvm/ranksvml2.pdf`.