Large-scale Linear RankSVM

Ching-Pei Lee
Department of Computer Science
National Taiwan University

Joint work with Chih-Jen Lin
Outline

1. Introduction
2. Our approach
3. Related Works
4. Experiments
5. Conclusions
Outline

1. Introduction
2. Our approach
3. Related Works
4. Experiments
5. Conclusions
Learning to rank is widely used in web search, recommendation system and online advertisement.

Mainly 3 categories of methods:

- **Pointwise**: Equivalent to regression
- **Pairwise**: Learn to classify preference pairs. Similar to classification. Ex: rankSVM
- **Listwise**: Try to directly optimize the measurement
Why Linear RankSVM?

We focus on pairwise method
- Pointwise methods do not consider different queries
- Listwise methods are slow, not significantly better

In pairwise methods, we focus on linear rankSVM
- rankSVM is a popular method extended from SVM
- Linear rankSVM is useful in quickly generating a baseline model

<table>
<thead>
<tr>
<th></th>
<th>Kernel</th>
<th>Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Good</td>
<td>Maybe worse</td>
</tr>
<tr>
<td>Training</td>
<td>Very slow</td>
<td>Fast</td>
</tr>
</tbody>
</table>
RankSVM (1/2)

- Training instances \( \{(y_i, x_i)\}_{i=1}^{l}, x_i \in \mathbb{R}^n \)
- \( y_i > y_j \): \( i \) is more preferred than \( j \)
- Goal: \( w^T x_i > w^T x_j, \forall y_i > y_j \)
- L1-loss linear rankSVM:
  \[
  \min_w \frac{1}{2} w^T w + C \sum_{y_i > y_j} \max \left( 0, 1 - w^T (x_i - x_j) \right)
  \]
- L2-loss linear rankSVM:
  \[
  \min_w \frac{1}{2} w^T w + C \sum_{y_i > y_j} \max \left( 0, 1 - w^T (x_i - x_j) \right)^2
  \]
$k$ different values for $y_i$. 2 scenarios:
- $k$ is small and fixed: data labeled by experts
- $k = O(l)$: labels are real numbers. ex: CTR

Each scenario has algorithms deal with it well but being poor in the other one

Our contribution: develop a algorithm that is fast for both scenarios
Difficulty and Possible Solution

- The main difficulty is the $O(l^2)$ terms in

$$\sum_{y_i > y_j} \max \left( 0, 1 - \mathbf{w}^T (\mathbf{x}_i - \mathbf{x}_j) \right),$$

solve it as a SVM problem costs $O(l^2 n)$

- Notice

$$\begin{cases} 
1 - \mathbf{w}^T \mathbf{x}_i + \mathbf{w}^T \mathbf{x}_j \leq 0 \\
1 - \mathbf{w}^T \mathbf{x}_j + \mathbf{w}^T \mathbf{x}_k \leq 0 
\end{cases} \Rightarrow 1 - \mathbf{w}^T \mathbf{x}_i + \mathbf{w}^T \mathbf{x}_k \leq 0$$

- A careful design can avoid redundant comparisons
Outline

1. Introduction
2. Our approach
3. Related Works
4. Experiments
5. Conclusions
$O(l^2)$ difficulty occurs in any optimization method that needs to evaluate function and gradient.

We solve L2-loss rankSVM using a trust region Newton method (TRON) by Lin and Moré (1999) for the differentiability.
Trust Region Newton Method

- A Newton-type method that iteratively minimizes a twice-differentiable function $f(w)$
- In the $t$-th iteration, given $w^t$ and a region $\Delta_t$, solve

$$
\min_s \quad q_t(s) \equiv \nabla f(w^t)^T s + \frac{1}{2} s^T \nabla^2 f(w^t) s, \|s\| \leq \Delta_t
$$

We then assign $w^{t+1} = w^t + s$

- Minimize by conjugate gradient (CG) method
- **Hessian-vector product**: one per CG iteration, thus is the bottleneck
Formulation (1/3)

\[ A \equiv \begin{bmatrix} \vdots \ & \cdots \ i \ & \cdots \ j \ & \cdots \ \vdots \end{bmatrix} = \begin{bmatrix} 0 \cdots 0 & +1 & 0 \cdots 0 & -1 & 0 \cdots 0 \end{bmatrix} \]

- If \( y_i > y_j \) then a row in \( A \) has that the \( i \)-th entry is 1, the \( j \)-th entry is -1

- Example:

\[
\begin{array}{c|ccc}
  i & 1 & 2 & 3 \\
  y_i & 2 & 3 & 1 \\
\end{array}
\]

\[
A = \begin{bmatrix}
  1 & 0 & -1 \\
-1 & 1 & 0 \\
  0 & 1 & -1 \\
\end{bmatrix}
\]
$D_w$ is a diagonal matrix:

$$(D_w)_{(i,j),(i,j)} \equiv \begin{cases} 
1 & \text{if } 1 - w^T(x_i - x_j) > 0, \\
0 & \text{otherwise.}
\end{cases}$$

$e \equiv [1, \ldots, 1]$, $X \equiv [x_1, \ldots, x_l]^T$
Chapelle and Keerthi (2010) use

\[ \nabla^2 f(w)v = v + 2C \left( X^T \left( A^T \left( D_w \left( A(Xv) \right) \right) \right) \right) \]

- \( A \) and \( D_w \) both have only \( O(l^2) \) non-zero elements

<table>
<thead>
<tr>
<th>Operation</th>
<th>( Xv )</th>
<th>( Av' )</th>
<th>( D_wv'' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>( O(ln) )</td>
<td>( O(l^2) )</td>
<td>( O(l^2) )</td>
</tr>
</tbody>
</table>

- The complexity is reduced from \( O(l^2n) \) to \( O(l^2 + ln) \)
- But \( O(l^2) \) may still be too large
\( A_w: \) exclude the rows in \( A \) with \( (D_w)_{(i,j),(i,j)} = 0 \)

\[
A_w^T A_w = A^T D_w A
\]

Thus \( \nabla^2 f(w)v = v + 2CX^T A_w^T A_w XVv \)

\( (A_w^T A_w)_{i,j} = \sum_s (A_w)_s,i (A_w)_s,j \)

\( (A_w)_s,i (A_w)_s,j = (i\text{-th entry}) \times (j\text{-th entry}) \text{ in row } s \)
Solution (2/5)

- **Example:**  \( A_w = \begin{bmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \)

- Each row of \( A_w \) contains only 2 non-zero elements, thus \((A_w)_{s,i}(A_w)_{s,j}\) can have the following values:
  - 1: \( i = j, (A_w)_{s,i} = 1 \) or \(-1\)
  - \(-1\): \( i \neq j, ((A_w)_{s,i},(A_w)_{s,j}) = (1,-1) \) or \((-1,1)\)
  - 0: otherwise

- Define \( SV \equiv \{(i,j) \mid (A_w)_{s,i}(A_w)_{s,j} = -1\} \), equivalent to \( \{(i,j) \mid (D_w)_{(i,j),(i,j)} = 1\} \)
Solution (3/5)

\[
(A_w^T A_w)_{i,j} = \begin{cases} 
|\{i \mid (i,t) \text{ or } (t,i) \in SV(w)\}| & \text{if } i = j, \\
-1 & \text{if } i \neq j, (i,j) \text{ or } (j,i) \text{ is in } SV(w), \\
0 & \text{otherwise}.
\end{cases}
\]

\[\downarrow\]

\[
(A_w^T A_w X v)_i = \sum_{j=1}^{l} (A_w^T A_w)_{i,j} (X v)_j \\
= (\|\{i \mid (i,t) \in SV(w)\}\| + \|\{i \mid (t,i) \in SV(w)\}\|) x_i^T v - \sum_{j: (i,j) \in SV(w)} x_j^T v - \sum_{j: (j,i) \in SV(w)} x_j^T v
\]
Define

\[ SV_i^+(w) \equiv \{ j \mid (j, i) \in SV(w) \} \]
\[ SV_i^-(w) \equiv \{ j \mid (i, j) \in SV(w) \} \]

and

\[ l_i^+(w) \equiv |SV_i^+(w)|, \quad \alpha_i^+(w, v) \equiv \sum_{j \in SV_i^+(w)} x_j^T v, \]
\[ l_i^-(w) \equiv |SV_i^-(w)|, \quad \alpha_i^-(w, v) \equiv \sum_{j \in SV_i^-(w)} x_j^T v. \]

Thus

\[ (A_w^T A_w X v)_i = (l_i^+(w) + l_i^-(w)) x_i^T v - \alpha_i^+(w, v) - \alpha_i^-(w, v) \]
Therefore,

\[
X^T A_w^T A_w X v
\]

\[
= X^T \left[ \left( l_i^+ (w) + l_i^- (w) \right) x_i^T v - \left( \alpha_i^+ (w, v) + \alpha_i^- (w, v) \right) \right]
\]

If we have the values of \( l_i^+ (w), l_i^- (w), \alpha_i^+ (w, v), \) and \( \alpha_i^- (w, v) \), Hessian-vector products can be calculated in \( O(ln) \) time.

But how to obtain these values?
Computing the Values

2 methods, both require sorting the data first

- Direct counting: costs $O(lk)$, $k$ is the number of different $y_i$
  - Works well when $k$ is small
  - When $k = O(l)$, $O(lk) = O(l^2)$
- Order-statistic tree: costs $O(l \log k)$
Airola et al. (2011) first calculate $l_i^+(w)$ and $l_i^-(w)$ by an order-statistic tree.
Solve L1-loss by cutting plane method.
Our procedure is extended from theirs: need to compute $\alpha_i^+(w, v)$ and $\alpha_i^-(w, v)$ in addition.
Order-statistic Trees (2/7)

\[ SV_i^+(w) \equiv \{ j \mid (j, i) \in SV(w) \} \]

\[ = \{ j \mid y_j > y_i, \mathbf{w}^T \mathbf{x}_j < \mathbf{w}^T \mathbf{x}_i + 1 \} \]

\[ = \{ j \mid y_j > y_i \} \cap \{ j \mid \mathbf{w}^T \mathbf{x}_j < \mathbf{w}^T \mathbf{x}_i + 1 \} \]

- Difficulty: both the order of \( y_i \) and \( \mathbf{w}^T \mathbf{x}_i \) are involved
- Traverse the data by the order of \( \mathbf{w}^T \mathbf{x}_i \), then the elements in \( \{ j \mid \mathbf{w}^T \mathbf{x}_j < \mathbf{w}^T \mathbf{x}_i + 1 \} \) are known
- Assume \( \mathbf{w}^T \mathbf{x}_1 \leq \cdots \leq \mathbf{w}^T \mathbf{x}_l \)
Only need to know \(|SV_i^+(w)|\): order statistics

Balanced binary search trees is suitable: \(\log k\) depth if \(k\) nodes, can be extended to order-statistic trees

For any \(i\), we arrange elements of \(\{j \mid w^T x_j < w^T x_i + 1\}\) in an order-statistic tree

Maintain 2 values in each node:

\[
\begin{cases}
\text{key} : y_i, \\
\text{size}(y_i) : \text{Number of instances in tree}(y_i)
\end{cases}
\]

Instances with the same \(y_i\) are put in the same node
Order-statistic Trees (4/7)

Example:

\[
\begin{array}{cccccc}
  i & 1 & 2 & 3 & 4 & 5 & 6 \\
y_i & 4 & 7 & 9 & 9 & 2 & 11 \\
\end{array}
\]

Assume \( \{j \mid w^T x_j < w^T x_1 + 1\} = \{1, 2, 3, 4, 5, 6\} \)

\((7, 6)\)

\((2, 2)\)  \((9, 3)\)

\((4, 1)\)  \((11, 1)\)

We store (key, size) in each node

\[l_1^+(w) = |\{j \mid y_j > 4, j \in \text{tree}(7)\}| + |\{j \mid y_j = 7 \text{ or } j \in \text{tree}(9)\}| + |\{j \mid y_j > 4, j \in \text{tree}(2)\}| = (6 - 2) + |\{j \mid y_j > 4, j \in \text{tree}(4)\}| = 4\]
Formalize the previous example by defining

\[
\text{Larger}(y, y_i) \equiv |\{j \mid y_j > y_i, j \in \text{tree}(y)\}|
\]

\[
= \begin{cases} 
0 & \text{if } y \text{ is a leaf, and } y \leq y_i, \\
\text{size}(y) & \text{if } y \text{ is a leaf, and } y > y_i, \\
\text{size}(y\text{'s right child}) & \text{if } y \text{ is not a leaf, and } y = y_i, \\
\text{Larger}(y\text{'s right child}, y_i) & \text{if } y \text{ is not a leaf, and } y < y_i, \\
\text{Larger}(y\text{'s left child}, y_i) & \text{if } y \text{ is not a leaf, and } y > y_i, \\
\text{size}(y) - \text{size}(y\text{'s left child}) & \text{if } y \text{ is not a leaf, and } y > y_i,
\end{cases}
\]

Thus \( l_i^+(w) = \text{Larger(\text{root of tree}, y_i)} \)

Cost is \( O(\log k) \)
When moving from $i$ to $i + 1$, maintain the tree by inserting the following instances into it:

$$\{ j \mid w^T x_i + 1 \leq w^T x_j < w^T x_{i+1} + 1 \}$$

- Find suitable leaf to insert and maintain balance both cost $O(\log k)$
- Update size of nodes traversed during insertion
Notice
\[\begin{align*}
    l_i^+(w) &= \sum_{j \in SV_i^+(w)} 1 \\
    \alpha_i^+(w, v) &= \sum_{j \in SV_i^+(w)} x_j^T v
\end{align*}\]

Analogue to size, store the following at each node
\[x_v(y) \equiv \sum_{j : j \in \text{tree}(y)} x_j^T v\]

Larger can then be extended to compute \(\alpha_i^+(w, v)\)
\(l_i^-(w)\) and \(\alpha_i^-(w, v)\) can be computed similarly

Complexity is now reduced from \(O(l^2 n)\) to \(O(ln + l \log l + l \log k)\)
Most balanced binary search trees involve some complicated operations to keep being balanced.

Simpler way: store keys in leaves $\Rightarrow$ need to know the number of different keys in advance.

Internal nodes: no keys thus no instances.

Size and $x_v$ are still defined in the same way.
Selection Trees (2/3)

- Example:
  
  \[
  \begin{array}{ccccccc}
  i & 1 & 2 & 3 & 4 & 5 & 6 \\
  y_i & 4 & 7 & 9 & 9 & 2 & 11 \\
  \end{array}
  \]

- Assume \( \{ j \mid w^T x_j < w^T x_1 + 1 \} = \{1, 2, 3, 4, 5, 6\} \)

\[
l_1^+(w) = |\{ j \mid y_j > 4\}| = 3 + 1 = 4
\]
Selection Trees (3/3)

\[
\text{Larger}(s) = \begin{cases} 
\text{Larger(\text{parent of } s)} + \text{size(\text{ sibling of } s)} & \text{if } s \text{ is the left child,} \\
\text{Larger(\text{parent of } s)} & \text{if } s \text{ is the right child,} \\
0 & \text{if } s \text{ is the root.}
\end{cases}
\]

\[
l^+_i(w) = \text{Larger(\text{the leaf node with key } = y_i)}
\]
## Related Works

<table>
<thead>
<tr>
<th>Work</th>
<th>Loss</th>
<th>Opt.</th>
<th>Values</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRSVM+ (Chapelle and Keerthi, 2010)</td>
<td>L2</td>
<td>Newton</td>
<td>direct</td>
<td>$O(lk)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>counting</td>
<td></td>
</tr>
<tr>
<td>Joachims (2006)</td>
<td>L1</td>
<td>cutting</td>
<td>direct</td>
<td>$O(lk)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>plane</td>
<td>counting</td>
<td></td>
</tr>
<tr>
<td>TreeRankSVM</td>
<td>L1</td>
<td>cutting</td>
<td>red-black</td>
<td>$O(l \log k)$</td>
</tr>
<tr>
<td>(Airola et al., 2011)</td>
<td></td>
<td>plane</td>
<td>tree</td>
<td></td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Our approach
3. Related Works
4. Experiments
5. Conclusions
Different Trees and Direct Counting

MSLR 30k ($k = 5$)

YAHOO set 1 ($k = 5$)

MQ2007-list ($k = l$)

MQ2008-list ($k = l$)

Direct counting: $O(lk)$

Tree: $O(l \log k)$
Different RankSVM Algorithms (1/2)

MLSR 30k \((k = 5)\)

Relative fun. val.

Pairwise accuracy

NDCG

YAHOO set 2 \((k = 5)\)
Different RankSVM Algorithms (2/2)

MQ2007-list ($k = 1$)

Relative fun. val.  

Pairwise accuracy

MQ2008-list ($k = 1$)
### Comparison With Pointwise Methods (1/3)

<table>
<thead>
<tr>
<th>Data set</th>
<th>L2-loss RankSVM</th>
<th></th>
<th>L1-loss SVR</th>
<th></th>
<th>L2-loss SVR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training time (s)</td>
<td>NDCG</td>
<td>Training time (s)</td>
<td>NDCG</td>
<td>Training time (s)</td>
<td>NDCG</td>
</tr>
<tr>
<td>MQ2007</td>
<td>0.5</td>
<td>0.5211</td>
<td>23.9*</td>
<td>0.4757*</td>
<td>0.5</td>
<td>0.5157</td>
</tr>
<tr>
<td>MQ2008</td>
<td>0.5</td>
<td>0.4571</td>
<td>3.4*</td>
<td>0.4153*</td>
<td>0.2</td>
<td>0.4450</td>
</tr>
<tr>
<td>MSLR 30k</td>
<td>1601.6</td>
<td>0.4949</td>
<td>461.6</td>
<td>0.4742</td>
<td>202.4</td>
<td>0.4946</td>
</tr>
<tr>
<td>YAHOO set 1</td>
<td>334.8</td>
<td>0.7619</td>
<td>10.8</td>
<td>0.7586</td>
<td>172.7</td>
<td>0.7650</td>
</tr>
<tr>
<td>YAHOO set 2</td>
<td>11.2</td>
<td>0.7519</td>
<td>47.6</td>
<td>0.7470</td>
<td>20.8</td>
<td>0.7578</td>
</tr>
</tbody>
</table>

*: Reached maximum iteration of LIBLINEAR

\[ k = 5 \text{ for all the 5 data sets} \]
Comparison With Pointwise Methods (2/3)

<table>
<thead>
<tr>
<th>Data set</th>
<th>L2-loss RankSVM</th>
<th>L1-loss SVR</th>
<th>L2-loss SVR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training time (s)</td>
<td>Pairwise accuracy</td>
<td>Training time (s)</td>
</tr>
<tr>
<td>MQ2007</td>
<td>1.3</td>
<td>70.35%</td>
<td>23.9*</td>
</tr>
<tr>
<td>MQ2008</td>
<td>0.5</td>
<td>82.70%</td>
<td>3.4*</td>
</tr>
<tr>
<td>MSLR 30k</td>
<td>1601.6</td>
<td>61.52%</td>
<td>65.4</td>
</tr>
<tr>
<td>YAHOO set 1</td>
<td>117.1</td>
<td>68.39%</td>
<td>2.4</td>
</tr>
<tr>
<td>YAHOO set 2</td>
<td>11.2</td>
<td>69.74%</td>
<td>3.3</td>
</tr>
<tr>
<td>MQ2007-list</td>
<td>38.7</td>
<td>80.67%</td>
<td>1.0</td>
</tr>
<tr>
<td>MQ2008-list</td>
<td>16.6</td>
<td>82.07%</td>
<td>1.1</td>
</tr>
</tbody>
</table>

*: Reached maximum iteration of LIBLINEAR
Comparison With Pointwise Methods (3/3)

<table>
<thead>
<tr>
<th>Data set</th>
<th>Random forest (40 trees)</th>
<th>GBDT (20 trees)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training time (s)</td>
<td>Pairwise accuracy</td>
</tr>
<tr>
<td>MQ2007</td>
<td>14.8</td>
<td>66.16%</td>
</tr>
<tr>
<td>MQ2008</td>
<td>2.3</td>
<td>80.36%</td>
</tr>
<tr>
<td>MSLR 30k</td>
<td>5102.1</td>
<td>63.76%</td>
</tr>
<tr>
<td>YAHOO set 1</td>
<td>1672.2</td>
<td>70.69%</td>
</tr>
<tr>
<td>YAHOO set 2</td>
<td>58.7</td>
<td>68.76%</td>
</tr>
<tr>
<td>MQ2007-list</td>
<td>606.0</td>
<td>78.78%</td>
</tr>
<tr>
<td>MQ2008-list</td>
<td>423.3</td>
<td>82.04%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data set</th>
<th>Random forest (1000 trees)</th>
<th>GBDT (1000 trees)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training time (s)</td>
<td>Pairwise accuracy</td>
</tr>
<tr>
<td>MQ2007</td>
<td>345.3</td>
<td>69.07%</td>
</tr>
<tr>
<td>MQ2008</td>
<td>52.0</td>
<td>82.60%</td>
</tr>
<tr>
<td>YAHOO set 2</td>
<td>1406.9</td>
<td>71.91%</td>
</tr>
</tbody>
</table>

8 cores are used
Outline

1. Introduction
2. Our approach
3. Related Works
4. Experiments
5. Conclusions
Conclusions

- Different algorithms share the same bottleneck on computing the loss term that contains $O(l^2)$ pairs
- Our method is efficient for both small and large $k$
- Our method is faster than all state of the art algorithms/implementations
- A public tool has been released