Large-scale training of linear ranking support vector machines

Antti Airola

University of Turku, Department of Information Technology
{firstname.lastname}@utu.fi
Background

- Learning to rank: learn from prior observations to predict orderings for sets of new data points
- Web search, recommender systems, decision making
- Pairwise criterion
  - Given two alternatives, predict which is 'better'
  - Proposed methods: RankBoost, RankNet, ... RankSVM
- Our contribution: improved computational efficiency for linear RankSVM training
Learning to rank

- Past observations
  - inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \quad \mathbf{x}_i \in \mathbb{R}^n$
  - utility scores $y_1, y_2, \ldots, y_m \quad y_i \in \mathbb{R}$

- Utility: a measure of 'goodness'

- Ranking: data points ordered according to utilities

- Learn $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ for predicting utilities for new inputs
  - Goal: rankings produced by sorting according to $f(\mathbf{x})$ should be close to the true ranking
Learning from scored data appears similar to regression, but... 

- only the ordering of the predictions matters!
- correct: \([1, 2, 4, 6]\), predicted: \([2, 4, 122, 1347]\)  
  \(\rightarrow 0\) pairwise mistakes
- correct: \([1, 2, 4, 6]\), predicted: \([1, 2, 4, 0]\)  
  \(\rightarrow 3\) pairwise mistakes: \((1,0), (2,0), (4,0)\)
Ranking support vector machine (RankSVM)

- Herbrich et al. (1999)
- minimizes (convex approximation of) the number of pairwise mistakes on training data
- regularization to avoid over-fitting
- excellent ranking performance in many real-world applications
Computational costs

- **Straightforward implementation**
  - Train SVM on pairs of data points
  - Poor scalability \( O(m^4) \) or worse

- **Improved method for linear RankSVM (Joachims, 2006)**
  - Subgradient optimization ("cutting plane method")
  - Model the pairs only implicitly, \( O(m \log(m)) \) scaling
  - Limitation: number of allowed utility levels assumed small constant (e.g. \{"good","bad"\}, \{1 . . 5\})
  - If not the case, \( O(m^2) \)
Contributions

- Generalize the method of (Joachims 2006)
- Efficient subgradient and loss computations using self-balancing binary search trees
- Guaranteed $O(m \log(m))$ scaling always
- Scales as far as the non-zero entries in data matrix fit memory
Regularized risk minimization

\[ w^* = \arg\min_w \left\{ \frac{1}{N} \sum_{y_i < y_j} L(w^T(x_j - x_i)) + \lambda w^T w \right\} \]

\[ L(z) = \max(0, 1 - z) \]

Non-differentiable, difficult to optimize

**Notation**

- \( L : \mathbb{R} \rightarrow \mathbb{R}^+ \): hinge loss
- \( x_i \in \mathbb{R}^n \): i:th training example
- \( y_i \in \mathbb{R} \): i:th label
- \( w \in \mathbb{R}^n \): linear predictor
- \( \lambda \in \mathbb{R}^+ \): regularization parameter
- \( N \): normalizer

Antti Airola

Large-scale training of linear RankSVM
Pairwise hinge loss

\[ y < y' \]

\[ z = w^T(x' - x) \]

\[ L(z) = \max(0, 1 - z) \]
A vector $\mathbf{v} \in \mathbb{R}^n$ is a subgradient of a convex function $f$ at $\mathbf{x}_0$ if

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \langle \mathbf{v}, \mathbf{x} - \mathbf{x}_0 \rangle$$

for all $\mathbf{x} \in \text{dom}(f)$.
Bundle optimization

Bundle method (a.k.a. cutting plane method)

Pseudocode:
1. beginning of iteration: current solution $\mathbf{w}_t$
2. compute a subgradient of the empirical risk at $\mathbf{w}_t$
3. modify a (cheap to solve) quadratic program with this information
4. solve the quadratic program to gain $\mathbf{w}_{t+1}$
5. If not termination criterion, continue to next iteration

Main computational bottleneck: subgradient computations
Subgradient of the empirical risk

\[ \sum_{i=1}^{m} (c_i - d_i) x_i , \]

where

\[ c_i = |\{ j : (y_i < y_j) \land (w^T x_i > w^T x_j - 1)\}| \]

\[ (d_i \text{ defined analogously}) \]

linear time, if we know \( c_i \) and \( d_i \)
What does it take to compute $c_1 \ldots c_m$?

$$c_i = |\{j : (y_i < y_j) \land (\mathbf{w}^\top \mathbf{x}_i > \mathbf{w}^\top \mathbf{x}_j - 1)\}|$$

Number of instances with larger true utility, but too small predicted utility compared to the $i$:th one.
Algorithm 1 (the obvious approach)

- for each pair of training instances
- compare the true and predicted scores
- $O(m^2)$ cost
Algorithm 2 (Joachims 2006)

- let there be $k$ possible true scores
- sort training data according to predicted scores
- iterate through sorted data $k$ times to compute $c_i, d_i$
- $O(km + m \log(m))$ cost
- worst case: $k \approx m \rightarrow O(m^2)$
Order statistics tree

Example: #nodes with KEY > 20
Algorithm 3 (proposed)

- sort training data according to predicted scores
- 2 passes through sorted data
- for each instance, one $O(\log(m))$ insertion and query to order statistics tree
- overall: $O(m \log(m))$
Experiments

Antti Airola

Large-scale training of linear RankSVM
Conclusion

- $O(m \log(m))$ time algorithm for linear RankSVM
- Applicable also for partial orderings (query structure in IR)
  - $O(mq) \rightarrow O(m \log(q))$ with $q$ being average query size
- Open source implementation:
  
  http://users.utu.fi/aatapa/RankSVM
Thank You!