CoFiRank
Maximum Margin Matrix Factorization for Collaborative Ranking
Appendix

Markus Weimer[*] **Alexandros Karatzoglou**[†] **Quoc Viet Le**[‡] **Alex Smola**[§]

1 Ranking Losses

The main body of the paper describes the procedures to compute the function value $l(f, y)$ and its gradient $\partial_f l(f, y)$ (the 'bundle') for the NDCG loss. This is the information needed by the bundle methods to minimize the empirical risk.

We can actually use Algorithm **??** in combination with any loss. This section will describe how to compute the loss value and its gradient efficiently for many ranking losses.

1.1 Regression

For least mean squared regression, the loss and the gradient are defined as

$$l(f, y) = \frac{1}{2}(f - y)^2 \tag{1}$$

$$\partial_f l(y, f) = (y - f) \tag{2}$$

1.2 Ordinal Regression

For ordinal regression, we would like $f_i - f_j < 0$ whenever $y_i < y_j$. Whenever this relationship is not satisfied, we incur a cost $C(i, j)$ for preferring item i to item j.

Denote by m_i the number of items j for which $y_j = i$. In this case, there are $m^2 - \sum_{i=1}^{n} m_i^2$ pairs y_i, y_j for which $y_i \neq y_j$. Normalizing by the total number of comparisons we may write the overall cost of the estimator as

$$\frac{1}{M} \sum_{y_i < y_j} C(y_i, y_j) \{f_i > f_j\} \text{ where } M = \frac{1}{2}\left[m^2 - \sum_i m_i^2\right]. \tag{3}$$

The loss can be defined as

$$l(f, y) = \frac{1}{M} \sum_{y_i < y_j} C(y_i, y_j) \max(0, 1 + f_i - f_j) \tag{4}$$

Now the goal is to find an efficient algorithm for obtaining the number of times when the individual losses are nonzero such as to compute both the value and the gradient of $l(f, y)$. The complication arises from the fact that observations x_i with label y_i may appear in either side of the inequality

[*]Telecooperation Group, TU Darmstadt, Germany, mweimer@tk.informatik.tu-darmstadt.de
[†]Department of Statistics, TU Wien, alexis@ci.tuwien.ac.at
[‡]Computer Science Department, Stanford University, Stanford, CA 94305, quoc.le@stanford.edu
[§]SML, NICTA, Northbourne Av. 218, Canberra 2601, ACT, Australia, alex.smola@nicta.com.au

depending on whether $y_j < y_i$ or $y_j > y_i$. This problem can be solved as follows: sort f in ascending order and traverse it while keeping track of how many items with a lower value y_j are no more than 1 apart in terms of their value of f_i. This way we may compute the count statistics efficiently. Algorithm 1 describes the details, generalizing the results of [**?**]. Again, its runtime is $O(m \log m)$, thus allowing for efficient computation.

Algorithm 1 $(l, g) = \mathrm{Ordinal}(f, y, C)$

input Vectors f and y score matrix C
output Loss $l := l(f, y)$ and gradient $g := \partial_f l(f, y)$
for $i = 1$ **to** n **do**
  $h_i = 0$ and $u_i = m_i$
end for
set $c = [f - \frac{1}{2}, f + \frac{1}{2}] \in \mathbb{R}^{2m}$ (concatenate the vectors)
compute $M = 0.5(m^2 - \|u\|_2^2)$
rescale $C \leftarrow C/M$
index = QuickSort(c)
initialize $l = 0$ and $g = \vec{0}$
for $i = 1$ **to** $2m$ **do**
  $j = \mathrm{index}(i) \bmod m$ and $z = y_j$
  if $\mathrm{index}(i) \leq m$ **then**
    for $k = 1$ **to** $z - 1$ **do**
      $l \leftarrow l - C(k, z) u_k c_j$
      $g_j \leftarrow g_j - C(k, z) u_k$
    end for
    $h_z \leftarrow h_z + 1$
  else
    for $k = z + 1$ **to** n **do**
      $l \leftarrow l + C(z, k) h_k c_{j+m}$
      $g_j \leftarrow g_j + C(z, k) h_k$
    end for
    $u_z \leftarrow u_z - 1$
  end if
end for

1.3 Preference Relations

In general, our loss may be described by means of a set of preference relations $j \succeq i$ for arbitrary pairs $(i, j) \in \{1, \dots m\}^2$ associated with a cost $C(i, j)$ which is incurred whenever i is ranked above j. This set of preferences may or may not form a partial or a total order on the domain of all observations. In these cases efficient computations along the lines of Algorithm 1 exist. In general, this is not the case and we need to rely on the fact that the set P containing all preferences is sufficiently small that it can be enumerated efficiently. The loss is then given by

$$\frac{1}{|P|} \sum_{(i,j) \in P} C(i, j) \{f_i > f_j\} \tag{5}$$

Again, the same majorization argument as before allows us to write a convex upper bound

$$l(f, y) = \frac{1}{|P|} \sum_{(i,j) \in P} C(i, j) \max [0, 1 + f_i - f_j] \tag{6}$$

$$\text{where } \partial_f l(f, y) = \frac{1}{|P|} \sum_{(i,j) \in P} C(i, j) \begin{cases} 0 & \text{if } f_j - f_i \geq 1 \\ f_i - f_j & \text{otherwise} \end{cases} \tag{7}$$

The implementation is straightforward, as given in Algorithm 2.

Algorithm 2 $(l, g) = \text{Preference}(f, P, C)$

input Vectors f, preference set P, score matrix C
output Loss $l := l(f, y)$ and gradient $g := \partial_f(f, y)$

initialize $l = 0$ and $g = \vec{0}$
while $(i, j) \in P$ **do**
  if $f_j - f_i < 1$ **then**
    $l \leftarrow l + C(i, j)(1 + f_i - f_j)$
    $g_i \leftarrow g_i + C(i, j)$ and $g_j \leftarrow g_j - C(i, j)$
  end if
end while

1.4 Expected Rank Utility

Similar to the NDCG ranking measure, Expected Rank Utility is a permutation-dependent and position-dependent ranking measure.

Definition 1 (Expected Rank Utility) *Under the assumptions on y and π as in Definition* **??** *the Expected Rank Utility (ERU) and its normalized variant (NERU) are defined as [**?**]*

$$\text{ERU}(\pi, y) = \sum_{i=1}^{k} 2^{\frac{1-\pi_i}{\alpha-1}} \max(y_i - d, 0) \text{ and } \text{NERU}(\pi, y) = \frac{\text{ERU}(\pi, y)}{\text{ERU}(\text{argsort}(y), y)}. \tag{8}$$

Here d is a "neutral" vote and α is the viewing halflife. This means that the top positions have an exponentially higher weight than positions lower on the list. In other words, d is a cutoff beyond which we stop caring about the particular object.

Due to the similarities between NDCG and NERU, the optimization procedure described for NDCG can be applied directly for NERU. However, we have to use a new setting for vectors a and b

$$a_i := 2^{\frac{1-i}{\alpha-1}} \text{ and } b_i := \max(y_i - d, 0) \tag{9}$$