Finding The Most Probable Ranking of Objects with Probabilistic Pairwise Preferences

Mikhail Parakhin
Microsoft Corp,
1 Microsoft Way
Redmond, WA 98052

Patrick Haluptzok
Microsoft Corp,
1 Microsoft Way
Redmond, WA 98052

Abstract

This paper discusses the ranking of a set of objects when a possibly inconsistent set of pairwise preferences is given. We consider the task of ranking objects when pairwise preferences not only can contradict each other, but in general are not binary - meaning, for each pair of objects the preference is represented by a pair of non-negative numbers that sum up to one and can be viewed as a confidence in our belief that one object is preferable to the other in the absence of any other information. We propose a probability function on the sequence of objects that includes non-binary preferences and evaluate methods for finding the most probable ranking for this model using it to rank results of a Microsoft On-line Handwriting Recognizer.

1 INTRODUCTION

The task of pairwise preference learning and ranking of objects has attracted a lot of attention within the machine learning literature [2] [3] [5]. It usually consists of two stages. In the first stage for the set of objects O one learns a *preference function* $h : O \times O \mapsto [0, 1]$, where the values of $h(o_i, o_j)$ closer to 1 indicate that object o_i is preferred to the object o_j, and closer to zero the opposite. It is important to note, that the function h does not in general induce a linear ordering - for example, it might be, that $h(o_1, o_2) = h(o_2, o_3) = h(o_3, o_1) = 1$ for different objects o_1, o_2, o_3. Therefore, the second stage is necessary, where the ranking of the objects is produced that agrees as much as possible with the learned preference function. That second stage is what we are going to discuss in this paper.

As an example, consider an on-line handwriting word recognizer [1]. The result of recognition is the list of possible alternatives with some additional information provided for each hypothesis (such as the raw scores from each of the differenent classifiers inside the recognizer, the confidences for every letter in the word, the unigram frequency of the word in the vocabulary, etc), as discussed in [1]. We may want to sort this list according to the probability of each answer being correct, so that even if the first hypothesis is incorrect, the correct one will likely be close in the list, and it will be easy for a user to make corrections by simply choosing the right hypothesis from the list. One way we could approach the task of sorting the answers is to train a pairwise classifier, which for every pair of answers in the list will tell us how probable it is that one is preferable to the other. In this paper we are going to discuss ways of using these preferences to find the most probable ranking of the list.

The standard solutions to the problem of ordering the set of objects using pairwise preferences include the Greedy Ordering algorithm [5], ordering by the number of wins [3] and simply running the QuickSort algorithm ignoring the possible non-transitivity of the preference function [2]. Of those three, only the greedy ordering assumes that the preferences are not in general binary. Performance bounds have been proved for these algorithms, however they all used an additive loss function of the form:

$$Loss = \sum_{i \neq j} w(o_i, o_j) \tag{1}$$

where $w(o_i, o_j) = 0$ if o_i is preferred over the o_j (written as $o_i \succ o_j$) in "ideal ranking" or "ground truth", and some positive value otherwise. The problem with this approach is that usually the values $h(o_i, o_j)$ are the outputs of some classification algorithm, and most of the time the "closeness" of the output to 0 or 1 can be interpreted as a measure of confidence in the result. In other words, if $h(o_i, o_j) = 1$, classification algorithm is absolutely certain that $o_i \succ o_j$. However in the loss function (1) ranking $o_j \succ o_i$ will incur at most an additive penalty of $w(o_j, o_i)$. Correspondingly, algorithms can rank o_j higher than o_i, even though the classification algorithm is absolutely certain that this is incorrect. In this paper we, using the assumptions of Bradley-Terry model [4] [9], formulate a probability function over the permutations of the objects that induces a multiplicative penalty for "disobeying" recommendations of the pairwise classifier. We then propose three different algorithms for finding the most probable ranking in this model - one reminiscent of the beam search approximation for dynamic programming tasks and two based on relaxation - and empirically evaluate performance/accuracy tradeoffs of the proposed algorithms on both synthetic and real-world data.

2 PROBLEM FORMULATION

We are given a set of n objects $O_n = \{o_i | i = 0, .., n{-}1\}$. For every pair of the objects $\{o_i, o_j\}$ we are given a *preference probability* $p_{ij} : p_{ij} \in [0..1], p_{ij} = 1 - p_{ji}$, which reflects the strength of the belief that object o_i is preferred over object o_j. As in the Bradley-Terry model, we assume that $p_{ij} \approx Prob(o_i \succ o_j | o_i, o_j)$, in other words we assume that the preference probability is close to the probability of one object being preferred to the other given only information about these two objects. We further assume, that p_{ij} are independent (as discussed in [9]).

We define the ranking function $r(i)$:

$$r(i) : \{r(i) = j | o_{r(0)} \succ o_{r(1)} \succ \ldots \succ o_{r(i-1)} \succ \succ o_j \succ o_{r(i+1)} \succ \ldots \succ o_{r(n-1)}\} \tag{2}$$

In other words, $r(i)$ denotes the index of the object assigned to rank i. The probability of the full ordering of the objects $o_0, \ldots, o_{n-1}$, defined by the ranking function $r_k : p(r_k) = p(o_{r_k(0)} \succ o_{r_k(1)} \succ \ldots \succ o_{r_k(n-1)})$, can now be defined as

$$p(r_k) = \frac{1}{z(\alpha)} \exp\left(\alpha \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \ln p_{r_k(i) r_k(j)}\right) \tag{3}$$

The unnormalized probability will then be

$$\hat{p}(r_k) = \exp\left(\alpha \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \ln p_{r_k(i) r_k(j)}\right) \tag{4}$$

and the partition function $z(\alpha) = \sum_k \hat{p}(r_k)$ where summation is taken over all the possible ranking functions (and thus all the possible orderings of the objects), and α is a positive free parameter. The most probable ranking then is defined by the ranking function with the maximal probability: $r_{\max} = \arg\max_k p(r_k)$ Obviously $r_{\max}$ is the same for all the positive values of α, so without loss of generality we will assume $\alpha = 1$.

Proposition 1. Let the ordering $o_{r_{\max}(0)} \succ \ldots \succ o_{r_{\max}(n-1)}$ be the most probable ranking of the objects in a set O_n. Then every continuous subordering $o_{r_{\max}(m)} \succ o_{r_{\max}(m+1)} \succ \ldots \succ o_{r_{\max}(m+l-1)}$ will be the most probable ordering of objects in a subset $O_l = \{o_{r_{\max}(m)}, o_{r_{\max}(m+1)}, \ldots, o_{r_{\max}(m+l-1)}\}$

Proof. Proof is omitted due to space constraints. □

It immediately follows from Proposition 1 that in the most probable ordering any pair of neighboring objects is always ordered in such a way that the object with higher preference probability is placed first:

$$o_{r_{\max}(0)} \succ \ldots \succ o_{r_{\max}(i)} \succ o_{r_{\max}(i+1)} \succ \ldots \succ$$
$$\succ o_{r_{\max}(n-1)} \Rightarrow p_{r_{\max}(i) r_{\max}(i+1)} \geq p_{r_{\max}(i+1) r_{\max}(i)}$$

3 FINDING THE MOST PROBABLE RANKING

In what follows for notational simplicity we are going to assume that all the preference probabilities $p_{ij} \neq 0.5$. The extension to the case when some $p_{ij} = 0.5$ is trivial.

3.1 GRAPH REPRESENTATION

Let's create a fully connected graph, where nodes are objects and the edges are directed from the object with higher preference probability to the object with the lower one. This

p_{ij}	o_0	o_1	o_2	o_3
o_0	0	0.7	0.2	0.4
o_1	0.3	0	0.1	0.8
o_2	0.8	0.9	0	0.3
o_3	0.6	0.2	0.7	0

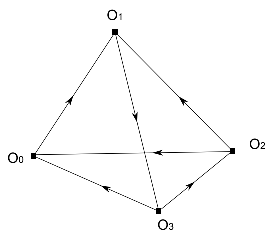

Figure 1. Example of the set of preference probabilities and the graph for $n = 4$.

graph is similar to the ones used in computing *Kemeny rankings* [6]. The difference is that we assume that we have a full set of p_{ij}'s, so the graph is always fully connected. The weights are computed and used differently, too.

We can now reformulate the task of finding the most probable ranking of the objects in graph terms:

- find all the paths in a directed graph, such that they pass through all the nodes and pass through each node exactly once (such path is called self-avoiding [10])
- for all such paths calculate the unnormalized probability (4) of the corresponding ordering and choose the most probable one

It is a well known fact that if A is an adjacency matrix of a directed graph, the number of paths of length n in this graph can be computed as the sum of all the elements of the matrix A^{n-1}. The (i, j) element of the matrix

A^n is the number of possible paths of length n between nodes i and j. In a similar way we can find all the paths of length n [8]. Simple modification of this algorithm will allow us to enumerate all the allowed paths and calculate unnormalized probabilities (4) of the corresponding orderings. We modify the adjacency matrix so that the (i, j) entry is a list of all allowed paths from node i to node j, where for each element in the list we also keep a partial unnormalized probability (4) - the unnormalized probability of the ordering, defined by the list element for the subset of objects included in this list element. In other words, for the path $o_0 \rightarrow o_1 \rightarrow \ldots \rightarrow o_m$ element of the list is $(\{o_0, o_1 \ldots, o_m\}, \hat{p}_m)$, where $\hat{p}_m = \exp\left(\sum_{i<j} \ln p_{ij}\right)$ The path is allowed if it passes through every node included in it exactly once. We now define an operation of multiplication of two list elements

$$\left(\{o_i, \ldots, o_m\}, \hat{p}_t\right) \times \left(\{o_l, \ldots, o_j\}, \hat{p}_f\right) =$$

$$= \begin{cases} \left(\{o_i, \ldots, o_m, \ldots, o_j\}, \hat{p}_{t+f}\right), \text{ if } m = l \text{ and there} \\ \qquad\qquad \text{are no repeating nodes} \\ \qquad\qquad \text{in } \{o_i, \ldots, o_m, \ldots, o_j\} \\ \emptyset \text{ otherwise} \end{cases}$$

where

$$\hat{p}_{t+f} = \hat{p}_t \hat{p}_f \exp\left(\sum_{l \in O_t} \sum_{s \in O_f} \ln p_{ij}\right),$$

and O_t is a set of indices of objects in the first path, O_f - in the second one. Result of multiplication of two lists is defined as a list of all the pairwise multiplications of their elements. The operation of addition of two lists is then simply their union. With operations defined in this way A^{n-1} will contain all the orderings with their unnormalized probabilities. All that's left is to cycle over them and find the most probable one.

This algorithm will always find the maximum probable ranking and it is faster than the naive one, but it is still exponential. To speed up calculations further we resort to approximation and apply technique used in approximate dynamic programming tasks : at every multiplication step t in every entry of the matrix we will keep only the d best sub-orderings - the d sub-orderings with the highest probability $\hat{p}_{t+1}$. This can be easily done using *min heap* datastructure [7].

3.2 RELAXATION-BASED METHODS

3.2.1 Probabilistic QuickSort

The main idea of Probabilistic QuickSort is very simple: at every step we will partition a set of objects into two subsets, such that all the objects in one of them (called the *prefered subset*) are likely to be preferred to the objects in the other one, and then recursively partition each part again. Please note, that despite the similar name, the following algorithm does not have much in common with the one, discussed in [2], where the standard QuickSort is analyzed in the setting when non-transitivity of the pairwise preferences is ignored.

Let O_p be a preferred subset of objects, and $O_n = O \setminus O_p$. At every partition then we want to choose O_p so as to maximize the probability of all the objects in O_p be preferred to objects in O_n. We also want the subset sizes to be approximately equal, so that at every partitioning step we are reducing the set of objects to rank by a factor of ~ 2. In other words we have the following optimization problem:

$$\min_{O_p, O_n} - \sum_{i \in O_p} \sum_{j \in O_n} \ln p_{ij} \tag{5}$$

$$\text{s.t.} \quad O_p \in O, \quad O_n = O \setminus O_p, \quad |O_p| \approx |O_n|$$

For every object o_i in a set we introduce an indicator variable $e_i \in \{-1, 1\}$. It is equal to 1 if the object is in the preferred subset and -1 otherwise. We then can reformulate the optimization problem (5) as

$$\min_E - \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} (e_i - e_j)(e_i - e_j + 2) \ln p_{ij} + \\ + (e_j - e_i)(e_j - e_i + 2) \ln p_{ji} \tag{6}$$

$$\text{s.t.} \quad \forall i \;\; e_i = \{-1, 1\}, \quad \sum_{i=0}^{n-1} e_i \approx 0$$

Equation (6) defines an integer optimization problem. The simplest way to solve it is by using relaxation: $e_i \in [-1, 1]$. We can simultaneously make it unconstrained by reparametrizing $e_i = \tanh cy_i$, where c is a parameter that controls the "degree of relaxation": the original integer problem is recovered with $c \rightarrow \infty$. After adding the regularization part, that would force the subsets' sizes to be approximately equal, and some simplification we arrive at the following optimization problem:

$$\min_X F = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} (\tanh cy_j - \tanh cy_i) \times \\ \times \left((\tanh cy_i - \tanh cy_j) \ln(p_{ij} p_{ji}) + 2 \ln \frac{p_{ij}}{p_{ji}}\right) + \\ + \lambda \left(\sum_{i=0}^{n-1} \tanh cy_i\right)^2 \tag{7}$$

where λ is a regularization parameter that defines a trade-off between "goodness" of separation and the difference between group sizes. We can now set all the y_i to zero and then minimize the functional (7) using plain gradient descent or any of the more sophisticated methods. After optimization we define $O_p = \{o_i : y_i \geq 0\},\ \ O_n = O \setminus O_p$.

3.2.2 Positional Ordering

In Probabilistic QuickSort we had to solve $O(n)$ optimization problems. We are now going to describe an alternative relaxation-based algorithm, where only one optimization problem has to be solved.

For every pair of object o_i,o_j we introduce an indicator variable $s_{ij} = \{0, 1\}$, $s_{ij} = 1 - s_{ji}$, which is equal to 1 if $o_i \succ o_j$ in the final ordering, and zero otherwise. Then the logarithm of an unnormalized probability (4) of any particular ordering defined by the set $\{s_{ij}\}$, can be rewritten as

$$\ln \hat{p} = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \ln \left(p_{ij} s_{ij} + p_{ji} s_{ji}\right)$$

However, we can not maximize this log probability directly, as s_{ij} are not independent - they should form a consistent set with the transitivity condition satisfied. To create a consistent set of s_{ij}, for every object o_i we introduce *position variable* $x_i \in (-\infty, \infty)$. The value of the position variable will correspond to the rank of the object in the ordering: the object with the highest x_i will be ranked the first, with the second highest - the second and so forth. Then we can formulate the following optimization problem:

$$\min_S - \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \ln \left(p_{ij} s_{ij} + p_{ji} s_{ji}\right) \qquad (8)$$

$$\text{s.t.} \quad \forall i, j \ \ s_{ij} = H(x_i - x_j), \ \ H(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0, \end{cases}$$

$$x_i \in (-\infty, \infty)$$

Solving this problem exactly would give us the exact most probable ranking, as this is just a reformulation of the task of maximizing the probability. The exact solution is difficult to find, however, as this is a non-linear non-convex optimization problem. In this paper we opt for approximation. We again introduce relaxation: instead of using step function $H(x)$, we can use sigmoid function $S(x) = (1 + \exp(-cx))^{-1}$, where c again is a parameter that controls the "degree of relaxation": and the original integer problem is recovered as $c \to \infty$. After introducing relaxation, we can formulate an unconstrained minimization problem:

$$\min_X - \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \ln \Big(p_{ij} S(x_i - x_j) + \\ + p_{ji}\big(1 - S(x_i - x_j)\big)\Big) \qquad (9)$$

4 EXPERIMENTS

In order to compare the proposed algorithms we created two datasets: one artificial and one based on real data. For the artificial dataset we simply generated the preference probabilities randomly from a uniform distribution independently for every pair of objects. There were sets from 5 to 40 objects in the artificial dataset. The second dataset was based on the real probabilities being extracted from the Microsoft On-Line Handwriting Recognizer [1]. The recognizer returns a list of up to 30 possible alternatives for an ink word, and for all pairs of alternatives the recognizer produces pairwise preference probabilities. Many of the ink words in the real data produce alternatives that are trivial to order - there is only one self-avoiding path in a graph which is discovered equally well by all the methods. In order to accentuate the differences, we ran the graph-based algorithm on the real data (approximately 36000 words) keeping the $d = 500$ most probable sub-orderings in every matrix element at every step and then chose 200 sets with the highest number of discovered self-avoiding paths.

As a baseline accuracy measure we have chosen a simple modification of the Greedy Ordering algorithm from [5][1]: for each object we calculate $p_i = \sum_{j=0,j\neq i}^{n-1} \ln p_{ij}$, then find i, corresponding to the highest p_i, postulate that object o_i is preferred to all the other objects, and then repeat this procedure with object o_i extracted from the set. The probability of the resulting ordering is denoted by P_g. We can now compare any other algorithm with this greedy approach using the average value of the *accuracy gain* $g = n^{-1}(\ln P(o_{r(0)} \succ \ldots \succ o_{r(n-1)}) - \ln P_g)$ as a comparison criteria.

Figure 2 shows the dependence of the accuracy gain and the algorithms' running time on the number of objects in a set. The position of the datapoints was perturbed slightly in the horizontal direction to make standard errors more visible. For this experiment we kept $d = 500$ sub-orderings at each element.

5 CONCLUSIONS

In this paper we studied the problem of finding the most probable ranking of the set of objects when preference probabilities are known for every pair of objects. We showed the connection between this problem and a problem in graph theory and proposed three algorithms for finding the most probable ranking. Evaluation on both synthetic and real-world datasets showed that none of the algorithms outperformed the others in all the situations and each one has it's strengths and weaknesses. That would suggest that it prob-

[1]Modifications are neccessary so that the algorithm would perform a valid greedy optimization of the functional (3).

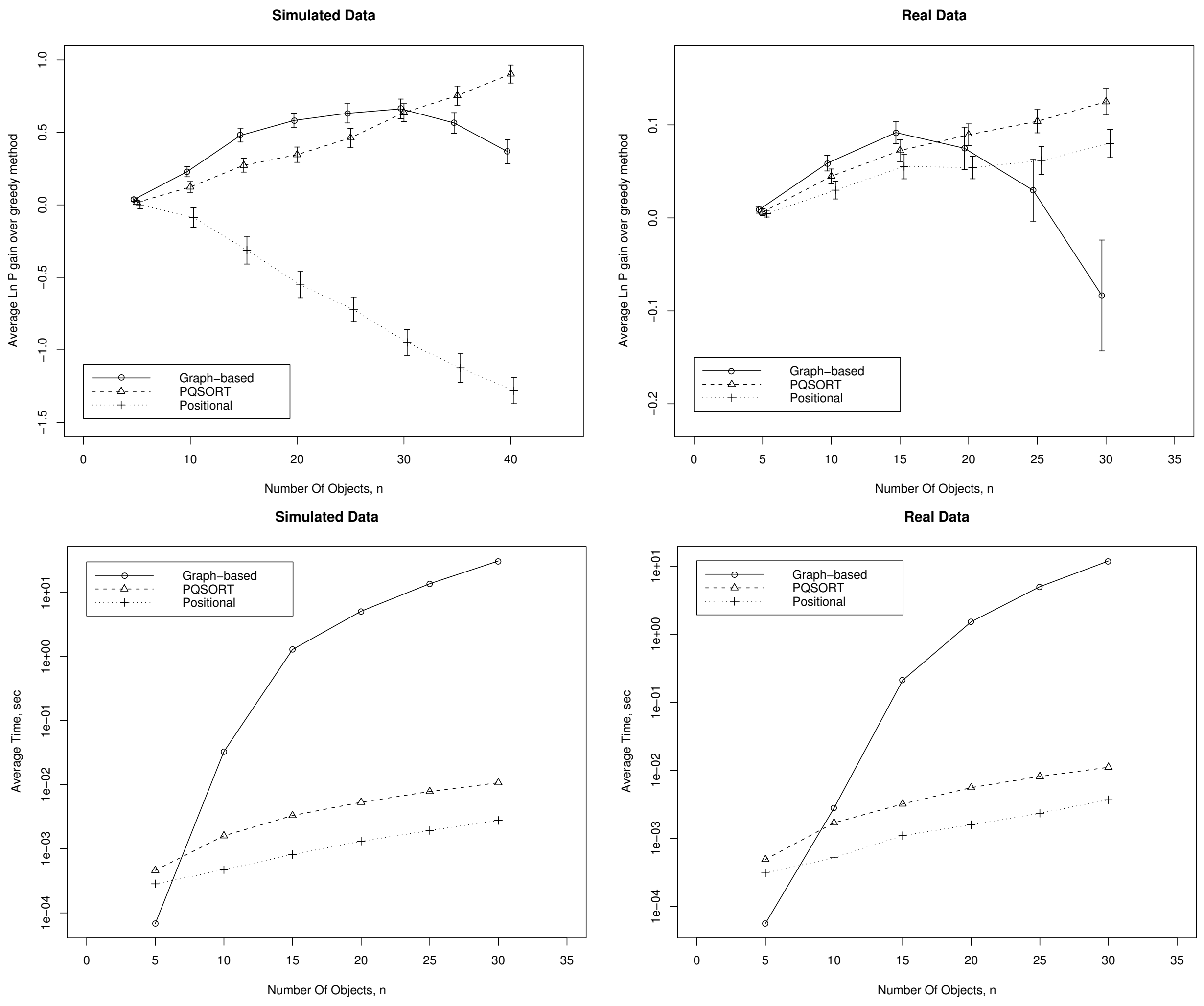

Figure 2. Average accuracy gain over the Greedy Ordering and running time for different algorithms.

ably makes sense to combine the algorithms to get optimal results.

References

[1] A. Abdulkader, J. A. Drakopoulos, and Q. Zhang. Comparative classifier aggregation. In *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition*, pages 156–159, Washington, DC, USA, 2006. IEEE Computer Society.

[2] N. Ailon and M. Mohri. An efficient reduction of ranking to classification. Technical report, NYU, 2007.

[3] M.-F. Balcan, N. Bansal, A. Beygelzimer, D. Coppersmith, J. Langford, and G. B. Sorkin. Robust reductions from ranking to classification. *Mach. Learn.*, 72(1-2):139–153, 2008.

[4] R. Bradley and M. Terry. The rank analysis of incomplete block method of paired comparisons. *Biometrika*, 39, 1952.

[5] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 451–457, Cambridge, MA, USA, 1998. MIT Press.

[6] V. Conitzer. Improved bounds for computing kemeny rankings. In *In In Proceedings of the 21st National Conference on Artificial Intelligence (AAAI*, pages 620–627. AAAI Press, 2006.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms. Second Edition.* The MIT Press, 2001.

[8] G. H. Danielson. On finding simple paths and circuits in a graph. *IEEE Transactions on Circuit Theory*, CT-15, 1968.

[9] T. Hastie and R. Tibshirani. Classification by pairwise coupling. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 507–513, Cambridge, MA, USA, 1998. MIT Press.

[10] J. Ponstein. Self-avoiding paths and the adjacency matrix of a graph. *Journal of SIAM*, 14, 1966.