A Practical Divide-and-Conquer Approach for Preference-Based Learning to Rank

Han-Jay Yang
Department of Computer Science
and Information Engineering
National Taiwan University
hanjayyang@gmail.com

Hsuan-Tien Lin
Department of Computer Science
and Information Engineering
National Taiwan University
htlin@csie.ntu.edu.tw

***Abstract*—Preference-based learning to rank (LTR) is a model that learns the underlying pairwise preference with soft binary classification, and then ranks test instances based on pairwise preference predictions. The model can be viewed as an alternative to the popular score-based LTR model, which learns a scoring function and ranks test instances based on their scores directly. Many existing works on preference-based LTR address the step of ranking test instances as the problem of weighted minimum feedback arcset on tournament graph. The problem is somehow NP-hard to solve and existing algorithms cannot efficiently produce a decent solution. We propose a practical algorithm to speed up the ranking step while maintaining ranking accuracy. The algorithm employs a divide-and-conquer strategy that mimics merge-sort, and its time complexity is relatively low when compared to other preference-based LTR algorithms. Empirical results demonstrate that the accuracy of the proposed algorithm is competitive to state-of-the-art score-based LTR algorithms.**

I. Introduction

The problem of learning to rank (LTR) arises in many applications ranging from web search to recommendation systems [1]–[3]. Given a list of items, the goal of LTR is to rearrange the items in a certain order such that the more relevant items are ranked before the less relevant ones. Because of its significance for vast applications, many different models are proposed to deal with the LTR problem [1]–[3].

There are two major categories of LTR models, namely *score-based* models and *preference-based* models. In score-based models, the learning algorithm in the training stage aims to produce a scoring function that maps each item to a real-valued score; then, the prediction algorithm produces the final ranking from the linear order induced from the scoring function. The training stage of score-based LTR models is in this sense similar to that of common regression models, which also map items to scores. Many works in tackling the LTR problem thus borrow the so-called pointwise ranking perspective from regression, such as PRanking [4] and large margin ordinal regression [5].

Nevertheless, score-based LTR models care about the goodness of the final ranking while regression models care about the accuracy of scores themselves. Such difference makes pointwise ranking less satisfactory in producing a decent final ranking. Many other score-based LTR models therefore try to optimize different ranking-related loss functions. The loss functions often depend on the pairwise or listwise relations among the whole ranking. For instance, the Kendal-Tau loss function calculates the number of mis-ranked item pairs in a ranking list and is considered in the pairwise ranking approaches like RankBoost [6], RankSVM [7]–[9], and RankNet [10]. Other loss functions are designed based on common metrics used in information retrieval such as NDCG [11] and MAP [12], and depend on the whole ranking order rather than the order of item pairs. Those listwise loss functions are the core of listwise LTR approaches. Some of the approaches like SVM-MAP [13] exploit a smoothed surrogate of the loss function, and others like AdaRank [14] and LambdaMART [15] directly optimize the non-convex objective by approximating the gradient during optimization.

Overall the pairwise and listwise approaches have delivered promising results [2], [16], boasting superior ranking performance and offering more versatility in dealing with different ranking metrics. Regardless of the approaches taken, however, score-based models are required to provide a pointwise (item-wise) score for an item. Such requirement makes optimizing against pairwise/listwise ranking objective difficult, leading to a usually more sophisticated training stage for pairwise/listwise score-based models.

Preference-based models have been considered in various works [17]–[19]. In preference-based models, instead of learning the scoring function for each particular item, a preference function over pairs of items is learned in the training stage. The preference function is then decoded to produce a ranking on a set of test items in the prediction stage. The decoding is generally done by minimizing the pairwise error (Kendall-Tau) subject to the preference function on the set of test items. Minimizing such pairwise error, however, is equivalent to a well-known NP-hard problem called weighted Min-FAST (weighted minimum feedback arc set on tournament graph) [20], [21].

Despite for some works that are based on heuristics [22], existing works on solving the weighted Min-FAST problem often focus on the theoretical perspective. For instance, [18] proves that a simple approach [17], which is of time complexity quadratic to the number of test instances, is a 2-approximation of the optimal solution for a special ranking task called bipartite ranking. [19], [23] make improvements by using a quick-sort-like approach to achieve 3-approximation within

sub-quadratic time complexity, and [24], [25] further achieve $(1+\epsilon)$-approximation within sub-quadratic time. Given the theoretical nature of the works, though many of them have discussed the possibility to employ preference-based LTR [19], [25], [26], few have yet to design algorithms that work well in practice or examine them properly on real-world data.

This work is dedicated to resolve the issue above. We design FUZZY-SORT, a divide-and-conquer approach for the prediction stage of preference-based LTR. FUZZY-SORT is inspired by the quick-sort routine within [19], [23], and employs a bottom-up routine to recursively sort items in the query list, actively querying more important pairwise preferences in the process. The divide-and-conquer scheme enables FUZZY-SORT to return a ranking list in sub-quadratic complexity. Experiments demonstrate that FUZZY-SORT is indeed more efficient than existing approaches for solving the Min-FAST task for preference-based LTR. In addition, experiments against several preference-based and score-based ranking algorithms on real-world data validate that FUZZY-SORT delivers competitive accuracy in most of the data sets. The results make FUZZY-SORT a first-hand choice for preference-based LTR and a promising approach for LTR problems.

II. BACKGROUNDS

We first consider the general notion of ranking. Given a set of N items numbered from 1 to N, a *ranking* of the items is a permutation $\pi \in S_N$, where S_N is the permutation group of order N, so the i-th ranked (or i-th relevant) item is $\pi(i)$. The goal of *learning to rank* (LTR) is to learn from training data a ranking model that is capable of producing accurate rankings for future unseen lists of items.

The process of LTR could generally be broken down into two stages. In the training stage, a set of items with indices $X = \{1, 2, \ldots, N\}$ and their corresponding feature vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ are given as input; also given are K queried subsets of items $X_k \subseteq X$ along with their true rankings π^*_k being a permutation of items in X_k for $k = 1, 2, \ldots, K$. Note that the true rankings usually correspond to the responses taken from different queries in real world applications, so it is possible for true rankings in different queried lists to disagree in some of the item orders. The task of a ranking algorithm in the training stage, using the given information, is to train a model capable of reconstructing a good ranking over an arbitrary given set of items.

In the prediction stage, again a (possibly different) set of items with indices $X' = \{1, 2, \ldots, N'\}$ and their corresponding feature vectors $\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_{N'}$ are given. The task is to produce a ranking π over X' that is as close to the (unknown) true ranking π^* as possible.

For the preference-based LTR framework, a *preference function* $h\colon \mathcal{X}^2 \to [0, 1]$, where $\mathcal{X}$ is the domain of feature vectors, is trained in the training stage. The preference function attempts to dictate the relative precedence of two given items softly. A larger $h(\mathbf{x}_i, \mathbf{x}_j)$ indicates that one would prefer $\mathbf{x}_i$ to be ranked before $\mathbf{x}_j$. The preference function h is then given as an "oracle" in the prediction stage, where the ranking is constructed from the oracle's judgment over pairwise preferences of the items.

One final note is that depending on how the preference function h is produced, the symmetric relation $h(\mathbf{x}_i, \mathbf{x}_j) = 1 - h(\mathbf{x}_j, \mathbf{x}_i)$ may not necessarily hold. Asymmetric preference functions can always be symmetrized by $h_{sym}(\mathbf{x}_i, \mathbf{x}_j) = \frac{h(\mathbf{x}_i, \mathbf{x}_j)}{h(\mathbf{x}_i, \mathbf{x}_j) + h(\mathbf{x}_j, \mathbf{x}_i)}$ if either $h(\mathbf{x}_i, \mathbf{x}_j)$ or $h(\mathbf{x}_j, \mathbf{x}_i)$ is non-zero, or $h_{sym}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2}$. We will drop the sym subscript and consider symmetric preference functions only in this work.

There are various different measures to evaluate the ranking performance [27]. Some popular ones are pairwise error [28], NDCG [11] and MAP [12]. For the scope of this paper, we consider the pairwise error given its close connection to preference-based LTR. For a given ranking π on N items, its pairwise error is simply the proportion of wrongly ordered pairs $L(\pi) = \frac{\sum_{i \prec j} \mathbb{1}(j \prec_\pi i)}{N(N-1)/2}$, where $\prec$ denotes the precedence specified in the true ranking π^*, and $\prec_\pi$ denotes the precedence induced from π.

Because the preference function h obtained from the training stage of preference-based LTR can be inconsistent, the prediction stage needs to solve an optimization problem that minimizes the pairwise error subject to h. Such a problem is equivalent to the min-FAST (minimum weighted feedback arc set on tournament graph) problem. Specifically, given a weighted tournament graph $G = (V, W)$ the Min-FAST problem seeks to find a permutation of vertices π such that the quantity $\sum_{i \prec_\pi j} W_{ji}$ is minimized, where $\prec_\pi$ denotes precedence induced by the permutation π. The prediction stage of preference-based LTR can simply be done by letting the nodes in graph G correspond to the items to be ranked, and $W_{ij} = h(\mathbf{x}_i, \mathbf{x}_j)$. Then, minimizing $\sum_{i \prec_\pi j} W_{ji}$ is the same as minimizing the pairwise error with respect to the soft predictions $h(\mathbf{x}_i, \mathbf{x}_j)$.

Theoretically, the weighted min-FAST problem is NP-hard to solve [23]. Thus, currently, approximation is required to realistically solve it. Two approaches on solving the problem are highly relevant to this work, as discussed below.

a) FAS-PIVOT*:* [23] showed that simply applying QUICK-SORT using $h(\mathbf{x}_i, \mathbf{x}_j)$ as a soft comparison function results in expected 3-approximation to the weighted min-FAST problem [23]. That is, if we perform QUICK-SORT multiple times with random pivots, and put i before j with probability $h(\mathbf{x}_i, \mathbf{x}_j)$, the expected pairwise error is no larger than 3 times the optimal error. The resulting FAS-PIVOT approach demonstrates that repetitively running a sorting algorithm of query complexity $O(N \lg N)$ on N items may lead to a practical way to solve min-FAST (and thus the prediction stage problem of preference-based LTR). Nevertheless, because the convergence rate of the approach is not clear, FAS-pivot potentially needs many iterations to converge, and it is yet to be studied whether FAS-pivot works reasonably well for preference-based LTR in practice.

b) SORT-BY-DEGREE*:* Long before FAS-PIVOT was proposed, SORT-BY-DEGREE [17] is a popular heuristic used

to solve the weighted min-FAST problem (for ranking). Formally, for each node i, we denote its net outgoing degree as

$$\delta_{net}(i) = \sum_{j \neq i} h(\mathbf{x}_i, \mathbf{x}_j) - h(\mathbf{x}_j, \mathbf{x}_i)$$

SORT-BY-DEGREE simply sorts the nodes by $\delta_{net}(\cdot)$ as the solution. [18] showed that Sorting-by-Degree is 2-optimal for the non-weighted, bipartite ranking version of the min-FAST problem, and [29] extended the theoretical result to the weighted, bipartite ranking version of the problem. The approach, however, requires calculating $\delta_{net}(i)$ for each node, which takes $O(N^2)$ time for N items, which can be slow if N is large.

III. PROPOSED APPROACH

As discussed, existing approaches for solving min-FAST suffer from efficiency. FAS-PIVOT may need potentially many iterations; SORT-BY-DEGREE requires quadratic amount of calculation. We hope to design an approach that is efficient enough for practice. Our idea lies somewhere in the middle of FAS-PIVOT and SORT-BY-DEGREE, combining the stronger suit of both approaches.

A. FUZZY-SORT

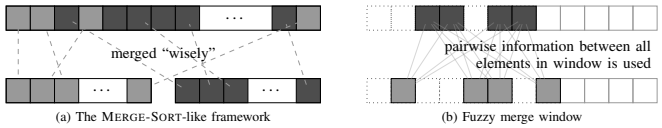
Consider first the MERGE-SORT alternative of the FAS-PIVOT approach. Instead of sorting the items like QUICK-SORT in a top-down fashion, we sort in a bottom-up order. Recall that MERGE-SORT recursively partitions the list of items to be ranked into two almost-equal-sized sublists until very short sublists are reached and trivially sorted. Two sorted sublists are then repeatedly merged from bottom to top, where the merging is done by making a single-direction pass on both-sublists to "zip" the items in proper order. During the pass, relation over two items would be determined by a single comparison, deciding their order in the merged list.

For the weighted min-FAST problems, the preference relations h between two items are some real values between 0 and 1. FAS-PIVOT treats those values probabilistically for comparison in QUICK-SORT. The probabilistic treatment results in instability, which is one of the reasons that FAS-PIVOT may need many iterations.

To solve the instability problem, we do not treat the values probabilistically. Instead, we borrow ideas from SORT-BY-DEGREE as part of a more stable merging subroutine. The subroutine is called FUZZY-MERGE because of its nature.[1] Consider in the merging step one is faced with two "roughly" sorted lists and want to merge them into another "roughly" sorted one. By "roughly" sorted we mean that most items in the lists are in order. It is then reasonable to assume that the (unknown) true item that should be ranked first is among the $W/2$-first items of both lists for some parameter W. We therefore use a fuzzy-merge window of size W that takes in the first $W/2$ items that are still available from both lists, and use SORT-BY-DEGREE to determine the item that should be placed first in the merged list. Once the locally-first item is determined, the window takes in the next item from list and iteratively selects the following winners one by one.

[1]We choose FUZZY here to hint that the merging is inexact in nature, not to be confused with anything related to the fuzzy logic.

The subroutine FUZZY-MERGE is shown in Algorithm 2, where the fuzzy window is denoted as the candidate set S which we will be selecting the locally-first item from. We start by initializing S with the first $W/2$ items from both lists. While we have not depleted both lists, we continuously select from S the locally-first item $\mathbf{x}$ using SORT-BY-DEGREE. The item $\mathbf{x}$ is then appended to the end of the current merged list A_M, and we update S to include the next possible item outside of S, depending on which list item $\mathbf{x}$ was taken from. At the end of FUZZY-MERGE, we would have perform $|A_L| + |A_R|$ selections of locally-first item from the candidate set S using SORT-BY-DEGREE, and the resulting merged list A_M is returned.



(a) The MERGE-SORT-like framework (b) Fuzzy merge window

Fig. 1. Shown in (a), FUZZY-SORT roughly follows the divide-and-conquer framework of MERGE-SORT. As demonstrated in (b), elements in the window for both lists are shaded, among which an elements is selected to be the "locally-first". Note that the elements in the window need not be continuous in a list, as elements in the middle could get picked prematurely by the sort-by-degree heuristic.

We hereby give our proposed approach for the prediction stage problem in Algorithm 1. The overall structure is pretty much the same with conventional MERGE-SORT, recursively splitting the lists in two, and fuzzily merging the returned (roughly) sorted lists.

Algorithm 1 FUZZY-SORT

Input: List of elements V, Oracle $h : V \times V \to [0, 1]$, window size W
Return: Reordering of elements in V
if $|V| \leq W/2$ **then**
  return V
end if
$W \leftarrow \min(W, |V|)$
$m \leftarrow \lfloor |V|/2 \rfloor$
$V_L \leftarrow \text{Fuzzy-Sort}(V[\,1 \ldots m\,], h, W)$
$V_R \leftarrow \text{Fuzzy-Sort}(V[\,m+1 \ldots |V|\,], h, W)$
return $\text{Fuzzy-Merge}(V_L, V_R, W)$

Algorithm 2 Fuzzy-Merge

1: **Input:** Two FUZZY-SORTed lists A_L, A_R, Window size W
2: **Return:** Merged list
3: $W_L \leftarrow \lfloor W/2 \rfloor$
4: $W_R \leftarrow \lceil W/2 \rceil$
5: $S \leftarrow \{A_L[1 \dots W_L]\} \cup \{A_R[1 \dots W_R]\}$
6: $A_L \leftarrow A_L[W_L + 1 \dots |A_L|]$
7: $A_R \leftarrow A_R[W_R + 1 \dots |A_R|]$
8: $A_M \leftarrow [\,]$
9: **while** $S \neq \phi$ **do**
10: $x \leftarrow \text{argmax}_{v \in S} \delta_S^{net}(v)$
11: $S \leftarrow S \backslash x$
12: $A_M.\text{Append}(x)$
13: **if** x came from A_L **then**
14: **if** $A_L \neq [\,]$ **then**
15: $S \leftarrow S \cup A_L.\text{Front}()$
16: $A_L.\text{Pop-Front}()$
17: **end if**
18: **else**
19: **if** $A_R \neq [\,]$ **then**
20: $S \leftarrow S \cup A_R.\text{Front}()$
21: $A_R.\text{Pop-Front}()$
22: **end if**
23: **end if**
24: **end while**
25: **return** A_M

For line 10 of Algorithm 2 (selection of the "locally-first" element in window), a naïve approach would need to examine the subgraph induced by all items in the window, in turn requiring time quadratic to window size. This is however not necessary. One can dynamically maintain $\delta_S^{net}(v)$ for any $v \in S$, and updating only when an item is added or removed from S in $O(W)$ operations. As there is only a fixed number of insertion and deletion of elements from S (linear to $|A_L| + |A_R|$), we see the time complexity and query complexity are both $O\left(W(|A_L| + |A_R|)\right)$ for subroutine FUZZY-MERGE, and consequently $O(W \cdot N \lg N)$ for FUZZY-SORT.

We will show in the experiment section that W is typically good enough at 50 or below, and this makes FUZZY-SORT a pretty realistic approach in terms of time and query complexity.

B. Philosophy behind the Algorithm

There are various ways to view our proposed algorithm under the preference-based LTR framework. Preference-based LTR has not yet gained much popularity as one of the many ranking schemes, mainly due to the instability on randomized approach and the unsatisfactory query complexity.

FUZZY-SORT can in a way be viewed as an attempt to trade some of the efficiency provided by a divide-and-conquer framework for the stability and performance provided by the deterministic heuristic, SORT-BY-DEGREE. The window size W is precisely the parameter that dictates the proportion of mixture between the two standpoints. At $W = 1$, FUZZY-SORT is exactly MERGE-SORT in the raw; at $W = N$, FUZZY-SORT is simply SORT-BY-DEGREE.

IV. EXPERIMENTS

This section will be roughly divided into two parts. In the first part we focus on the prediction stage optimization problem, i.e. where FUZZY-SORT stands among other previously proposed methods to solve the min-FAST problem for preference-based ranking; this will mainly be done on an artificially generated network (of item preference relation). In the second part we experiment and demonstrate the performance of FUZZY-SORT on real data sets, and provide comparison with other ranking algorithms, both preference-based and score-based.

For all of the following experiments and algorithms involved, we use pairwise ranking error as the evaluation measure. All algorithms employed in the experiment also has pairwise ranking error as their optimization objectives.

A. Comparison of Prediction Schemes on Artificial Data

We claimed through various perspective that FUZZY-SORT is efficient in fully utilizing the allotted queries to produce good ranking, and we shall verify first under a more idealistic setting that it is indeed so.

We adopt the Bradley-Terry-Luce (BTL) model [30], [31] for our artificial data generation. In the BTL model, each item is assumed to possess an (unknown) "true" score s_i that represents on a unified scale how high it should be ranked in a score-based model. Based on the score of items, the result of comparison between two items i and j is considered a Bernoulli variable with success probability related to the logistic of score difference, i.e. :

$$P\left[i \prec j\right] = \text{logistic}\left(c \cdot (s_i - s_j)\right) = \frac{1}{1 + \exp\left(c(s_j - s_i)\right)}$$

Though somewhat idealistic (as it may not always be the case that pairwise preferences in real data follows from a given total order), in the case where we want to obtain a total ranking, BTL is a natural model to describe pairwise preference relations generated from such ranking.

Thus, we have the item scores randomly uniformly generated over a designated interval, and use such scores as ground truth in the BTL model. To ensure the noise is "fixated" in the generated data, and precision of preference output does not get progressively better by querying a particular relation multiple times, the preference output corresponding to a particular pair of elements is fixed upon first query. The number of items is set to be 50000, while the logistic coefficient c is set so that $c(s_{max} - s_{min}) = 0.8$ [2]; In other words, the noise that arises for a single query is actually quite noticeable. The preference output given by oracle is the result of averaging a random number (no more than 15) of queries.

[2] As a matter of fact, the relative performance between different prediction algorithms was not greatly affected by the BTL model parameters. i.e. The plot stays similar even if we tweak the parameters, therefore we only show result under a fixed set of parameters here.

We compare FUZZY-SORT with several other counterparts being FAS-PIVOT, averaged-MERGE-SORT[3], a Markov chained based approach (RANK-CENTRALITY) by Negahban et al. [22], and the sort-by-degree heuristic in raw. Note the latter two algorithms query all quadratic pairs of preference relations before execution, while the previous three have performance that scale with the number of queries they are allotted to make. Therefore, To compare how each algorithm scales as they are given more "computational resource", we let FUZZY-SORT take the largest merge-window it could use under allotted amount of queries, while FAS-PIVOT and averaged-MERGE-SORT average over maximum number of iterations allowed.

Each algorithm is ran 10 times on the data set. The mean and standard deviation of the pairwise error is shown in Figure 2, plotted against number of queries used.

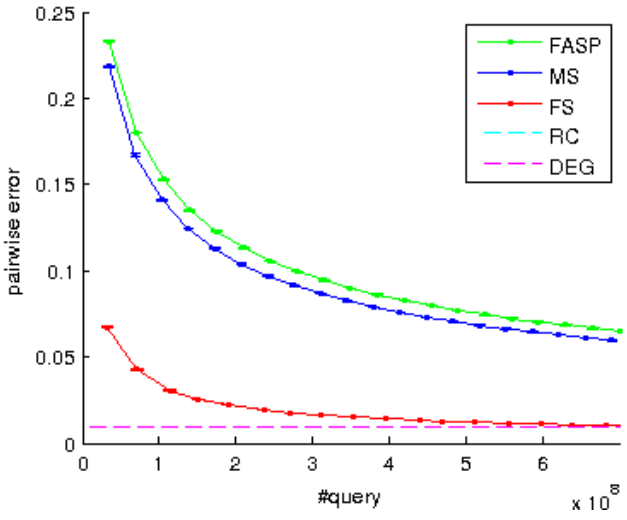


Fig. 2. Performance in pairwise error under fixed number of total query the accuracy of RANK-CENTRALITY (RC) and SORT-BY-DEGREE (DEG) are plotted as a dashed line as they require a quadratic amount (more than 10^9) of queries, out of the x-axis bound of the plot. The two dashed lines actually overlap with each other. Error bars are also plotted but are slightly too small to be distinguishable.

As can be seen in Figure 2, FUZZY-SORT is dominant when compared to FAS-PIVOT and averaged-MERGE-SORT. Not only does FUZZY-SORT converges much quicker to good accuracy, the end of the curves also hint that at convergence FAS-PIVOT and average-MERGE-SORT is unlikely to reach as low a pairwise error as Fuzzy-Sort has, at least not before an excessive amount of extra iterations.

Compared with RANK-CENTRALITY and naïve SORT-BY-DEGREE which both uses full information of the pairwise relations (that is, more than 10^9 queries), FUZZY-SORT can reach comparable pairwise error in significantly less queries. Also at convergence, we see FUZZY-SORT essentially reaches what RANK-CENTRALITY and SORT-BY-DEGREE can achieve.

Note that the information of running time is omitted here as all these methods above are implemented such that the time complexity is the basically proportional to the query complexity, with the slight exception of RANK-CENTRALITY which has a constant factor for doing matrix transition several times.

B. Performance on Real Data

In this section we shall move to some real world data sets to compare our methods with fellow ranking algorithms.

Data Sets: We incorporate several data sets for testing. *bike*, *cadata*, *cpusmall* are originally real world regression data sets from UCI repository [32] taken as a single-query ranking data sets [33]–[35]. *MQ2007*, *MQ2008* are LTR data sets from the "Million Query Track" of TREC 2007 and TREC 2008 [36], [37], which consist of queries with preference label from 0 through 2. *YahooLTR1*, *YahooLTR2* are large-scale LTR data sets taken from Yahoo! Learning to Rank Challenge [38], both containing queries with preference label from 0 through 4.

In summary, in viewing the results shown in Table II, it is worth keeping in mind that *bike*, *cadata*, *cpusmall* are data sets with long query lists and a wide range of relevance score; *MQ2007*, *MQ2008* are data sets with pretty short query lists and a small range of preference labels; *YahooLTR1*, *YahooLTR2* are large-scale data sets with large amount of items and features. The detailed data statistics are shown in Table I.

Compared Methods: We compare several state-of-the-art score-based ranking algorithms against FUZZY-SORT, along with other preference-based methods.

The three score-based algorithms we shall include are RankSVM [7], RankBoost [6] and LambdaMART [15]. Being the winner of Yahoo! Learning to Rank Challenge in 2010, LambdaMART is arguably the most powerful score-based ranking algorithm now, capable at handling different kind of optimization objectives and has shown promising performance across numerous data sets. Rank-SVM is as well one of the representative score-based linear model proposed in as early as 2002, still constantly being compared with in many survey of ranking approaches. RankBoost is another ranking model with some history that stems from the idea of AdaBoost, but has rather a non-linear scoring function.

For all preference-based ranking algorithm, we perform very simple and effortless training to form the oracle. We do simple subsampling over all item-pairs with different preference label, among which a random forests [39] is trained as the binary classification model. The number of sampled edges is around 1 to 25 times the number of items, with the ratio being *smaller* for large data sets (e.g. 50000 for *MQ2008*, 200000 for *YahooLTR1*). Random forests are trained with 50 trees and a bagging ratio of 0.5.

Other ranker specifics are given below in more detail:

- FUZZY-SORT: Models are trained with a window size of 50.
- FAS-PIVOT: The result of the average from 50 iterations.

[3] We perform MERGE-SORT by comparisons whose result is probabilistically decided by item preference relations, and average the result of several separate MERGE-SORT as in FAS-PIVOT.

- Averaged-MERGE-SORT: The result of the average from 50 iterations.
- RANK-CENTRALITY: The distribution after 20 iterations is considered as output. For RANK-CENTRALITY algorithm, iterations beyond 20-th hardly affects the performance so 20 iterations are in fact very sufficient (in terms of convergence).
- Rank-SVM: We use Joachim's implementation in package SVM$^{\text{Rank}}$ [7], with parameter C chosen from: $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\}$.
- RankBoost: We use the implementation in RankLib [40] with a small tweak in the code to support optimization against pairwise ranking error.
- LambdaMART: As with RankBoost we use the implementation in RankLib with a manual tweak to support pairwise ranking error.[4] LambdaMART models are trained with 1000 trees and learning rate 0.1.

The result of the experiment is shown in Table II; entries that represent best statistics for a data set are marked in boldface, where closeness are determined under one-tailed t-test at 5% significance level. The running time for each algorithm is shown in Table III. For preference-based algorithms, the time required for training stage where a random forest is build and the time required for prediction stage where the ranking is generated from the random forest are separately listed. For score-based algorithms, since prediction stage generally takes minimal time (several seconds to a minute top depending on algorithm), we simply omit the prediction entries of score-based algorithm in the table.

1) FUZZY-SORT *versus other preference-based prediction schemes:* As shown in Table II, FUZZY-SORT essentially outperform FAS-PIVOT and averaged-MERGE-SORT in all data sets. RANK-CENTRALITY, based on the idea of random-walk of a Markov Chain, while requiring much more computational resource and shown to be as-good-as in our artificial experiments, does not really outshine FUZZY-SORT in real data sets. In most data sets FUZZY-SORT and RANK-CENTRALITY has really close performance, with FUZZY-SORT seemingly winning slightly in general.

Factoring in the time efficiency (to be discussed) for each preference-based algorithm, it is probably safe to say FUZZY-SORT is the best choice of all preference-based algorithms tested.

2) FUZZY-SORT *versus score-based ranking algorithms:* Compared against conventional score-based ones, FUZZY-SORT performs reasonably well and is largely better than Rank-SVM and RankBoost, while falling slightly behind LambdaMART in most data sets. LambdaMART is undeniably good across all data sets; at least with the random forest model we are using as the binary prediction model, preference-based methods does not beat LambdaMART on most data sets.

[4]We initially gave the gbm package [41] in R a try due to its supporting pairwise ranking error by default, but the implementation in gbm seems somewhat inferior to that of RankLib

However the pairwise accuracy deficit is not that big on most data sets, and the training time on larger data sets is heavily in favor of FUZZY-SORT.

3) Running time comparisons: From Table III, it is evident that out of the four tested preference-based algorithms, FUZZY-SORT outperform all others in efficiency. Under conditions where a short amount of time could be allowed before generating the prediction, FUZZY-SORT proves to be a very good choice in the class of preference-based algorithms, able to generate prediction mostly in within a minute or two. Most importantly, contrary to what was believed that preference-based algorithms have terrible prediction stage efficiency, the prediction time of FUZZY-SORT actually comes pretty close to that of LambdaMART, meaning FUZZY-SORT is hardly impractical if LambdaMART is being used.

Due to the training model we choose, the training time for preference-based algorithms is directly proportional to the number of features in the data and the number of sampled pairs. Arguably the training time is not most satisfiable, but it could be tuned down when necessary (while sacrificing accuracy) and is generally consistent and controllable.

Out of all algorithms tested, score-based methods across the board show some difficulties dealing with longer query lists (*cadata*,*YahooLTR1*), as their computational complexity still mostly scale with the amount of pairs in given query lists. Rank-SVM appears to have the most robust training efficiency, having a fast training time even on larger data sets (*YahooLTR1,YahooLTR2*). LambdaMART on the other hand is much slower in general and quickly becomes painfully slow as the data size gets larger, as could be seen in *cadata,YahooLTR1,YahooLTR2*.

One last thing to note though, is that for the training time for preference-based algorithms is currently decided by the time to train a random forest on sampled preference pairs. So the training time listed in the table, while it is indeed the time we spent to train our preference-based oracle, is by no means tied with given preference-based algorithms. It is always possible to choose a different model to train as an oracle, with different training efficiency and prediction performance. Here the method we choose is and is intended to be somewhat more simplistic, in part to demonstrate the possibility of choosing just choosing a simple model to train against given preference pairs in the preference-based learning framework.

V. CONCLUSION

For the past few years, there have been scattered works that endorsed preference-based LTR; most however focused on the theoretical side, and it would seem that preference-based LTR is a framework more of theoretical interest rather than practical use.

In this work we show otherwise by demonstrating preference-based LTR could achieve promising performance in ranking scenario. Most importantly, in FUZZY-SORT the existing prediction-stage efficiency issue is met with active-sampling done in a recursive fashion, thereby making the

TABLE I
DATA STATISTICS.

Data Set	#samples	#features	#query	#relevance-level
bike	7380	16	1	594
cadata	10000	8	1	3279
cpusmall	4000	12	1	54
MQ2007	13652	46	1700	3
MQ2008	2874	46	800	3
Yahoo1	165660	700	6983	5
Yahoo2	103174	700	3798	5

TABLE II
PERFORMANCE ON REAL DATA SETS IN PAIRWISE ERROR. COMPARES DIFFERENT LEARNING ALGORITHMS, NAMELY FUZZY-SORT(FZS), RANK-CENTRALITY(RC), FAS-PIVOT(FASP), AVERAGED-MERGE-SORT(MERGE), RANK-SVM(RSVM), RANK-BOOST(BOOST), LAMBDA-MART(LMART) ON VARIOUS DATA SETS. SOME DATA SETS DETAILS ARE ALSO SHOWN. THE COLUMNS #SAMP, #FEA, #QRY, #LVL GIVES NUMBER OF SAMPLES, NUMBER OF FEATURES, NUMBER OF QUERIES, NUMBER OF RELEVANCE LEVEL, RESPECTIVELY.

Data Set	FZS	FASP	MERGE	RC	RSVM	BOOST	LMART
bike	.0748±.0001	.0907±.0009	.0862±.0003	.1028	.1756	.2486	**.0586**
cadata	.2251±.0001	.2386±.0009	.2251±.0003	.2287	**.1643**	.2902	.2158
cpusmall	.0939±.0004	.0972±.0004	.0981±.0003	.0920	.1050	.1657	**.0775**
MQ2007	**.0889±.0004**	.0902±.0009	.0910±.0006	**.0886**	.0932	.1532	**.0887**
MQ2008	**.0553±.0008**	.0568±.0018	**.0562±.0018**	.0576	.0562	.1400	.0652
Yahoo1	.1782±.0004	.1849±.0005	.1805±.0005	.1805	.2031	.2142	**.1626**
Yahoo2	.1497±.0002	.1545±.0003	.1522±.0003	.1476	.1587	.1544	**.1409**

TABLE III
TRAINING/PREDICTION TIME REQUIRED FOR ALGORITHMS IN SECONDS WITH A DEBIAN LINUX ON INTEL(R) XEON(R) CPU E5 AT 2.40GHZ, 16CORES. PREDICTION TIME FOR SCORE-BASED ALGORITHM IS OMITTED. [*] EARLY-STOP DUE TO NON-TERMINATION IN GIVEN TIME.

Data Set	preference-based Algorithms					RSVM	BOOST	LMART
	Training	FZS	FASP	MERGE	RC			
bike	210	110	294	312	1932	2460	2244	26
cadata	211	167	305	347	3181	14763	2556	11823
cpusmall	264	52	97	109	667	162	351	108
MQ2007	288	106	1330	1438	18	4	153	624
MQ2008	127	2	17	17	3	2	33	108
Yahoo1	3195	131	1183	1320	222	569	21611	86400*
Yahoo2	2831	60	691	619	122	376	675	15602

prediction-phase efficiency much more reliable even against long query lists.

Our experimental results hints the possibility of an easier training phase under preference-based LTR framework. The classification task on pairwise preference could be done with accuracy, also number of pairs needed to train a decent oracle is surprisingly small.

In summary our work shines in the context that pairwise information are more readily accessible, or a pointwise model is somewhat harder to train. One could consider the analogy in real life that ever so often comparing between two items (e.g. which topic interests you more?) are easier than giving them an actual score (e.g. how interested are you toward this topic?). Such results are also more likely to be consistent as pointwise estimations are inevitably prone to relative shift in score. For these reasons preference-based LTR could be the favorable choice in such situations, and FUZZY-SORT is a solid, practical way to carry out preference-based predictions.

ACKNOWLEDGMENT

The work arises from the Master's thesis of the first author [29]. We thank Profs. Yuh-Jye Lee, Shou-De Lin, the anonymous reviewers and the members of the NTU Computational Learning Lab for valuable suggestions. This work is partially supported by the Ministry of Science and Technology of Taiwan via the grant MOST 103-2221-E-002-148-MY3.

REFERENCES

[1] T.-Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

[2] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 129–136.

[3] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the web," in *Proceedings of the 10th international conference on World Wide Web*, ser. WWW '01. New York, NY, USA: ACM, 2001, pp. 613–622. [Online]. Available: http://doi.acm.org/10.1145/371920.372165

[4] K. Crammer and Y. Singer, "Pranking with ranking," in *Advances in Neural Information Processing Systems 14*. MIT Press, 2001, pp. 641–647.

[5] A. Shashua and A. Levin, "Ranking with large margin principle: Two approaches," pp. 937–944, 2002.

[6] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *The Journal of machine learning research*, vol. 4, pp. 933–969, 2003.

[7] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 133–142.

[8] ——, "Training linear svms in linear time," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 217–226.

[9] R. Herbrich, T. Graepel, and K. Obermayer, "Large margin rank boundaries for ordinal regression," *Advances in Neural Information Processing Systems*, pp. 115–132, 1999.

[10] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 89–96.

[11] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.

[12] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[13] Y. Yue, T. Finley, F. Radlinski, and T. Joachims, "A support vector method for optimizing average precision," in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '07. New York, NY, USA: ACM, 2007, pp. 271–278. [Online]. Available: http://doi.acm.org/10.1145/1277741.1277790

[14] J. Xu and H. Li, "Adarank: a boosting algorithm for information retrieval," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 391–398.

[15] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao, "Ranking, boosting, and model adaptation," *Tecnical Report, MSR-TR-2008-109*, 2008.

[16] T.-y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li, "Letor: Benchmark dataset for research on learning to rank for information retrieval," in *In Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 2007.

[17] W. W. C. R. E. Schapire and Y. Singer, "Learning to order things," in *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference*, vol. 10. MIT Press, 1998, p. 451.

[18] M.-F. Balcan, N. Bansal, A. Beygelzimer, D. Coppersmith, J. Langford, and G. B. Sorkin, "Robust reductions from ranking to classification," *Machine learning*, vol. 72, no. 1-2, pp. 139–153, 2008.

[19] N. Ailon and M. Mohri, "Preference-based learning to rank," *Machine Learning*, vol. 80, no. 2-3, pp. 189–211, 2010.

[20] M. R. Garey and D. S. Johnson, *Computers and intractability*. wh freeman, 2002, vol. 29.

[21] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.

[22] S. Negahban, S. Oh, and D. Shah, "Iterative ranking from pair-wise comparisons." in *Advances in Neural Information Processing Systems*, 2012, pp. 2483–2491.

[23] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: Ranking and clustering," *Journal of the ACM*, vol. 55, no. 5, pp. 23:1–23:27, 2008. [Online]. Available: http://doi.acm.org/10.1145/1411509.1411513

[24] C. Kenyon-Mathieu and W. Schudy, "How to rank with few errors," in *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM, 2007, pp. 95–103.

[25] N. Ailon, "Active learning ranking from pairwise preferences with almost optimal query complexity." in *Advances in Neural Information Processing Systems*, 2011, pp. 810–818.

[26] J. Fürnkranz and E. Hüllermeier, "Pairwise preference learning and ranking," in *Machine Learning: ECML 2003*. Springer, 2003, pp. 145–156.

[27] C. D. Manning and P. Raghavan, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1.

[28] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, pp. 81–93, 1938.

[29] H.-J. Yang, "A practical divide-and-conquer approach for preference-based learning to rank," Master's thesis, National Taiwan University, 2014.

[30] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, pp. 324–345, 1952.

[31] R. D. Luce, *Individual choice behavior: A theoretical analysis*. Courier Dover Publications, 2005.

[32] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[33] H. Fanaee-T and J. Gama, "Event labeling combining ensemble detectors and background knowledge," *Progress in Artificial Intelligence*, pp. 1–15, 2013. [Online]. Available: [WebLink]

[34] B. Kelly Pace, "Sparse spatial autoregressions," 1997.

[35] M. Revow, "comp-activ dataset."

[36] J. Allan, B. Carterette, J. A. Aslam, V. Pavlu, B. Dachev, and E. Kanoulas, "Million query track 2007 overview," DTIC Document, Tech. Rep., 2007.

[37] B. Carterette, V. Pavlu, H. Fang, and E. Kanoulas, "Million query track 2009 overview." in *TREC*, 2009.

[38] O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview." in *Yahoo! Learning to Rank Challenge*, 2011, pp. 1–24.

[39] S. Bochkanov, "Alglib," www.alglib.net.

[40] V. Dang, "Ranklib."

[41] G. Ridgeway, "Generalized boosted regression models," *Documentation on the R Package gbm, version 1· 5*, vol. 7, 2006.