Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Multi-Prototype Label Ranking with Novel Pairwise-to-Total-Rank Aggregation

Mihajlo Grbovic*
Yahoo! Labs, USA
mihajlo@yahoo-inc.com

Nemanja Djuric*
Temple University, USA
nemanja.djuric@temple.edu

Slobodan Vucetic
Temple University, USA
slobodan.vucetic@temple.edu

Abstract

We propose a multi-prototype-based algorithm for online learning of soft pairwise-preferences over labels. The algorithm learns soft label preferences via minimization of the proposed soft rank-loss measure, and can learn from total orders as well as from various types of partial orders. The soft pairwise preference algorithm outputs are further aggregated to produce a total label ranking prediction using a novel aggregation algorithm that outperforms existing aggregation solutions. Experiments on synthetic and real-world data demonstrate state-of-the-art performance of the proposed model.

1 Introduction

In the Label Ranking setup, the input space is defined by a feature vector $\mathbf{x} \in \mathcal{X}$, and the output is defined as a ranking of labels $\pi \in \Pi$ (e.g., $\pi = (3, 1, 4, 2)$). Given a sample from the underlying distribution $D = \{(\mathbf{x}_n, \pi_n), n = 1, ..., N\}$, where $\mathbf{x}_n$ is a d-dimensional feature vector and π_n is a vector containing either a total or a partial order of a finite set $\mathcal{Y}$ of L class labels, the goal is to learn a model that maps $\mathbf{x}$ to a total label order $\mathcal{L} : \mathcal{X} \rightarrow \Pi$.

This problem has received a lot of attention in the machine learning community and has been extensively studied [Dekel *et al.*, 2003; **?**; **?**; **?**; **?**]. For a survey of recent Label Ranking algorithms see [Gärtner and Vembu, 2010]. Unlike standard problems of classification and regression, Label Ranking is a complex learning task, which involves prediction of strict label-order relations, rather than single values. It differs from the Learning to Rank setup [Cao *et al.*, 2007], where a feature vector is formed from query features and corresponding document features, and the output is the relevance score. It also differs from the Collaborative Filtering setup [Zhou *et al.*, 2008], where the input space is a set of scores over some of the labels, and the output is a set of scores or rankings of the remaining labels.

There are many practical applications in which the objective is to learn exact label preference of an instance in a form of a total order. For example, in the task of document categorization [Boley *et al.*, 1999] where it is very likely that a document belongs to multiple topics (e.g., science, entertainment, news, etc.) multi-label classification has traditionally been used. However, one might not be interested only in distinguishing between relevant and irrelevant topics for a specific document, but also in learning how to find a total order of the topics by relevance.

Personalization and targeting is emerging as an important and practically relevant field. Current trend is that publishers aim at personalizing their content (such as news articles, stories, advertisements) at the fine-grain level of individual user, or at least at the level of a user group [Grbovic *et al.*, 2012]. One way of achieving this is through learning user preferences over content categories or topics (e.g., sports, finance, politics, etc.) in form of total ranking π. Since the user reading history, such as the counts of reading sessions per topic, is directly correlated with the user preferences, one way of inferring ranking π is to sort the counts. However, new users that have just sign up with the publisher have no activity log, and the publisher must look elsewhere for predictive data sources that are indicative of π. Popular cold-start heuristics include randomly suggesting content, or suggesting content popular among a new user's age group or gender [Park and Chu, 2009], and usually fail to initially personalize beyond that. Label Ranking allows us to learn a mapping $\mathbf{x} \rightarrow \pi$, from existing users for which information $\mathbf{x}$ is available and π is known, where $\mathbf{x}$ is a vector created from user's sign-up information (e.g. age, gender, location) and/or third party information (e.g., occupation, hobbies, political views, religion). Since the preferences may change over time, it is important to have a label ranking model with online learning capabilities, such that it can be updated with daily activities of a large user population.

We propose a soft multi-prototype algorithm (SMP-Rank) for learning pairwise label preferences, attractive because of an intuitive online learning process, ease of implementation, and an ability to support various ranking structures, including complete and incomplete rankings, bipartite and multipartite ranking as well as arbitrary pairwise preferences. SMP-Rank belongs to a class of Learning Vector Quantization [Kohonen, 1990] prototype-based learning algorithms, that have previously been shown useful in classification [Seo and Obermayer, 2003] and regression [Grbovic and Vucetic, 2009] tasks. We derive its adaptation for the Label Ranking task. In addition, we propose a new algorithm for aggregation of arbi-

* These authors have contributed equally to this work.

trary pairwise preferences into a total ranking. The preference aggregation is a well-known problem in preference learning, where numerous methods have been proposed [Cohen *et al.*, 1999; **?**; **?**]. It is challenging, since the pairwise preferences may not be transitive. For instance, if we have three labels i, j and k, label ranking algorithm might prefer i over j, j over k, and finally k over i, thus inducing a cycle. The proposed aggregation algorithm consistently outperforms competing algorithms in this task as the number of labels L grows.

There are several existing approaches to Label Ranking. Pairwise Comparison method (PW) [Hüllermeier *et al.*, 2008] trains $L \cdot (L-1)/2$ classifiers, one for each pair of labels, and aggregates the pairwise predictions into a total order using Soft-Borda Count. Log-Linear Label Ranking (Lin-LL) method [Dekel *et al.*, 2003] learns mappings $f_k : \mathcal{X} \to \mathbb{R}$ for each label $k = 1, .., L$, to minimize the number of ranking errors. Each $f_k(\cdot)$ is modeled as a linear combination of base ranking functions, such that $f_i(\mathbf{x}) > f_j(\mathbf{x})$ if $\mathbf{x}$ prefers label i over j. The final rank is produced by sorting the scores $f_k(\cdot)$.

On the other hand, instance-based Mallows (IB-Mal) [Cheng *et al.*, 2009] and Plackett-Luce (IB-PL) [Cheng *et al.*, 2010] methods use the k-Nearest Neighbor algorithm to locate neighbors in the feature space, and rely on the Mallows [Mallows, 1967] or the Plackett-Luce [Plackett, 1975] probabilistic ranking models to combine neighbors' label rankings into a single prediction. The instance-based methods have been shown to outperform the competitors. However, their success comes at a large cost associated with both memory and time. Firstly, they require that the entire training data set is stored in memory, which can be very costly or even impossible in resource-constrained applications. Secondly, the Mallows and Plackett-Luce parameters are not learned during training, instead they are estimated at prediction time for each test instance separately, making it hard to achieve real-time predictions in web-based applications such as ad targeting.

There are several main justifications for using a prototype-based algorithm in Label Ranking. One is that it allows a clear mechanism for designing a prototype-based ranker on a fixed prototype budget. Unlike classification, where methods such as condensing [Hart, 1968] have been proposed for nearest neighbor classifiers, there is no clear alternative for building budget-based nearest neighbor and Parzen window algorithms for Label Ranking. Another justification is that the proposed SMP-Rank has low training cost with linear time and constant memory scaling with training size. In particular, gradient descent-based models, such as the proposed SMP-Rank, are most practical when it comes to needing to update the model with millions of examples every day, without having to store them. This makes it attractive for online learning applications.

There is one more property that distinguishes the proposed algorithm from the competition. Most of the existing algorithms are designed to learn from label orderings that form a ranking, either complete or partial top-k rankings. In order to be able to learn from arbitrary pairwise preferences that cannot form a ranking (e.g., $1 \succ 5, 2 \succ 1, 3 \succ 4$, where $\succ$ is read as "is preferred over") most of them need to improvise by imputing missing comparisons, for instance, based on conditional insertion probabilities [Lu and Boutilier, 2011]. SMP-Rank, however, handles such incomplete data directly, without having to estimate the missing information.

We conducted thorough empirical evaluation on synthetic and real-world data, with various data and label set sizes. The results illustrate the usefulness of the proposed methodology in the label ranking task, and also in the task of pairwise-to-total-rank aggregation, where the new aggregation algorithm outperforms the existing solutions.

2 Preliminaries

In the label ranking scenario, a single instance, described by a d-dimensional vector $\mathbf{x}$, is associated with a total order of assigned class labels, represented as a permutation π of the set $\{1, ..., L\}$, where L is the number of available class labels. We define π such that $\pi(i)$ is the class label at the *i*-th position in the order, and $\pi^{-1}(j)$ is a position of the y_j class label in the order. The permutation can also describe incomplete ranking $\{\pi(1), ..., \pi(k)\} \subset \{1, ..., L\}, k < L$.

In our approach we represent the label ranking for each instance $\mathbf{x}$ in the following way. Instead of the total order, we use a zero-diagonal preference matrix $\mathbf{Y}$, where $\mathbf{Y}(i,j) > \mathbf{Y}(j,i)$ if label y_i is preferred over label y_j and $\mathbf{Y}(i,j) + \mathbf{Y}(j,i) = 1$, for $i, j \in \{1, ..., L\}$. A value of $\mathbf{Y}(i,j)$ which is close to 1 is interpreted as a strong preference that y_i should be ranked before y_j. Similarly, uncertain (soft) preferences can be modeled by using values lower than 1. For example, indifferences (ties) are represented by setting $\mathbf{Y}(i,j) = \mathbf{Y}(j,i) = 0.5$. Given potentially incomplete permutation π, the preference matrix consistent with it can be formed in many ways. A typical approach is to assign $\mathbf{Y}(i,j) = 1$ and $\mathbf{Y}(j,i) = 0$ if $y_i \succ y_j$ and $\mathbf{Y}(i,j) = \mathbf{Y}(j,i) = 0$ in case of incomparable or missing class label preferences. This representation allows us to work with complete and top-k label rankings, bipartite and multipartite label rankings, as well as with arbitrary pairwise preferences and indifferences.

2.1 Evaluation Metrics

Let us assume that N historical observations are collected in a form of a data set $D = \{(\mathbf{x}_n, \mathbf{Y}_n), n = 1, ..., N\}$. The objective in all scenarios is to train a ranking function $\mathcal{L} : \mathbf{x}_n \to \hat{\pi}_n$ that outputs a total label order.

In the Label Ranking scenario, to measure the degree of correspondence between true and predicted rankings for the n-th example, π_n and $\hat{\pi}_n$ respectively, it is common to use the Kendall's tau distance $d_n = \mid \{(y_i, y_j) : \pi_n^{-1}(y_i) > \pi_n^{-1}(y_j) \wedge \hat{\pi}_n^{-1}(y_j) > \hat{\pi}_n^{-1}(y_i)\} \mid$. To evaluate a label ranking model, the label ranking loss on the data set D is defined as the average normalized Kendall's tau distance,

$$loss_{LR} = \frac{1}{N} \sum_{n=1}^{N} \frac{2 \cdot d_n}{L \cdot (L-1)}. \quad (1)$$

Note that the measure simply counts the number of discordant label pairs and reports the average over all considered pairwise rankings. Given the preference matrix representa-

tion, assuming binary predictions $\hat{\mathbf{Y}}_n$, we can rewrite (1) as

$$loss_{LR} = \frac{1}{N}\sum_{n=1}^{N}\frac{\parallel \mathbf{Y}_n - \hat{\mathbf{Y}}_n \parallel_F^2}{L\cdot(L-1)}, \quad (2)$$

where $\parallel \cdot \parallel_F$ is the Frobenius norm, and entries for which $\mathbf{Y}_n(i,j) = \mathbf{Y}_n(j,i) = 0$ are skipped over. For models with soft label preference predictions $\hat{\mathbf{Y}}_n$, e.g., $\hat{\mathbf{Y}}_n(i,j) = 0.7, \hat{\mathbf{Y}}_n(j,i) = 0.3$, loss (2) can be interpreted as a soft version of (1).

In the proposed methodology, we learn the mapping $\mathcal{L}: \mathbf{x}_n \rightarrow \pi_n$ in two stages. In the learning stage, function $f: \mathbf{x}_n \rightarrow \mathbf{Y}_n$ is learned via minimizing (2). In the aggregation stage, given the model predictions in a form of $\mathbf{Y}_n$, the total order prediction π_n is computed using a preference aggregation $h: \mathbf{Y}_n \rightarrow \pi_n$.

3 SMP-Rank algorithm

The proposed method is completely defined by a set of K prototypes $\{(\mathbf{m}_k, \mathbf{Q}_k), k = 1, ..., K\}$, where $\mathbf{m}_k$ is a d-dimensional vector in input space, and $\mathbf{Q}_k$ is the corresponding pairwise preference matrix.

The starting point in the design of SMP-Rank is to introduce the probability $\mathbb{P}(k \mid \mathbf{x})$ of assigning observation $\mathbf{x}_n$ to the k-th prototype that is dependent on their (Euclidean) distance. Let us assume that the probability density $\mathbb{P}(\mathbf{x}_n)$ can be described by a mixture model,

$$\mathbb{P}(\mathbf{x}_n) = \sum_{k=1}^{K}\mathbb{P}(\mathbf{x}_n \mid k)\cdot\mathbb{P}(k), \quad (3)$$

where K is the number of prototypes, $\mathbb{P}(k)$ is the prior probability that a data point is generated by the k-th prototype, and $\mathbb{P}(\mathbf{x}_n \mid k)$ is the conditional probability that the k-th prototype generates particular data point $\mathbf{x}_n$. Let us represent the conditional density function $\mathbb{P}(\mathbf{x}_n \mid k)$ with the normalized exponential form $\mathbb{P}(\mathbf{x}_n \mid k) = \theta(k)\cdot\exp(f(\mathbf{x}_n, \mathbf{m}_k))$ and consider a Gaussian mixture with $\theta(k) = (2\pi\sigma_p^2)^{-1/2}$ and $f(\mathbf{x}_n, \mathbf{m}_k) = -\|\mathbf{x}_n - \mathbf{m}_k\|^2/2\sigma_p^2$. We assume that all prototypes have the same standard deviation σ_p and the same prior, $\mathbb{P}(k) = 1/K$. Given these assumptions, using the Bayes' rule we can write the assignment probability as

$$g_{nk} \equiv \mathbb{P}(k \mid \mathbf{x}_n) = \frac{\exp(-\|\mathbf{x}_n - \mathbf{m}_k\|^2/2\sigma_p^2)}{\sum_{u=1}^{K}\exp(-\|\mathbf{x}_n - \mathbf{m}_u\|^2/2\sigma_p^2)}. \quad (4)$$

For compactness we introduced the following notation, $g_{nk} = \mathbb{P}(k \mid \mathbf{x}_n)$.

To derive a cost function for SMP-Rank, we use the prediction model that assigns data point $\mathbf{x}_n$ to the prototypes probabilistically and predicts based on the weighted average of the prototype preference matrices. Assuming such a model we can write the posterior probability $\mathbb{P}(\mathbf{Y} \mid \mathbf{x})$ as

$$\mathbb{P}(\mathbf{Y} \mid \mathbf{x}_n) = \sum_{k=1}^{K}\mathbb{P}(k \mid \mathbf{x}_n)\cdot\mathbb{P}(\mathbf{Y} \mid k) = \sum_{k=1}^{K} g_{nk}\cdot e_{nk}, \quad (5)$$

where we introduced $e_{nk} = \mathcal{N}(\mathbf{Y}_n - \mathbf{Q}_k, \sigma_y^2)$.

The mixture model assumes the conditional independence between $\mathbf{x}_n$ and $\mathbf{Y}$ given k, $\mathbb{P}(\mathbf{Y} \mid \mathbf{x}_n, k) = \mathbb{P}(\mathbf{Y} \mid k)$. For $\mathbb{P}(k \mid \mathbf{x}_n)$ we assume the Gaussian distribution from (4). For the probability of generating a preference matrix $\mathbf{Y}$ by prototype k, $\mathbb{P}(\mathbf{Y} \mid k)$, we also assume Gaussian error model with mean $(\mathbf{Y} - \mathbf{Q}_k)$ and standard deviation σ_y. The resulting cost function can be written as the negative log-likelihood,

$$loss_{smp} = -\frac{1}{N}\sum_{n=1}^{N}\ln\sum_{k=1}^{K}\mathbb{P}(k \mid \mathbf{x})\cdot\mathcal{N}(\mathbf{Y} - \mathbf{Q}_k, \sigma_y^2). \quad (6)$$

It is important to observe that, after proper normalization, (6) reduces to (2) if examples are assigned to prototypes deterministically. Therefore, it can be interpreted as its soft version. If prototype matrices $\mathbf{Q}_k$ consisted of only 0 and 1 entries (hard label preferences) (6) further reduces to (1).

The objective is to estimate the unknown model parameters, namely prototype positions $\mathbf{m}_k$ and their preference matrices $\mathbf{Q}_k, k = 1, ..., K$. This is done by minimizing the cost function $loss_{smp}$ with respect to these parameters. If online learning capability is a requirement, one can use the stochastic gradient descent method and obtain the learning rules at the t-th iteration by calculating derivatives $\partial loss_{smp}^t/\partial\mathbf{m}_k$ and $\partial loss_{smp}^t/\partial\mathbf{Q}_k$ for $k = 1, ..., K$. This results in following rules for the n-th training example,

$$\begin{aligned}\mathbf{m}_k^{t+1} &= \mathbf{m}_k^t - \alpha(t)\frac{(L_n - e_{nk})\cdot g_{nk}}{L_n}\frac{(\mathbf{x}_n - \mathbf{m}_k)}{\sigma_p^2}\\ \mathbf{Q}_k^{t+1} &= \mathbf{Q}_k^t - \alpha(t)\frac{e_{nk}\cdot g_{nk}}{L_n}\frac{(\mathbf{Y}_n - \mathbf{Q}_k)}{\sigma_y^2},\end{aligned} \quad (7)$$

where $L_n = \sum_{k=1}^{K} g_{nk}\cdot e_{nk}$. The resulting model has training time complexity of $\mathcal{O}(NKL)$. Note that it is straightforward to cast the optimization into EM framework following the procedure from [Weigend *et al.*, 1995].

At iteration $t = 0$, the procedure starts by selecting first K training points as initial prototypes. If any $\mathbf{Q}_k$ prototype preference matrix obtained in such manner contains empty elements, they are replaced with 0.5 entries, as the corresponding labels will initially be treated equally. Then, prototype parameters are learned using the iterative procedure (7). Learning is terminated when the loss function $loss_{smp}$ stops improving (e.g., changes by less than 10^{-5}).

During inference, rank prediction $\hat{\pi}_u$ of unlabeled data point $\mathbf{x}_u$ is determined as an aggregated weighted average of the prototype preference matrices $\mathbf{Q}_k$, $\hat{\pi}_u = h(\hat{\mathbf{Y}}_u)$, where $\hat{\mathbf{Y}}_u = (\sum_{k=1}^{K} g_{uk}\cdot\mathbf{Q}_k)$ is the aggregated preference matrix, and $h(\cdot)$ is an aggregation function presented in Section 4.

An important issue to be addressed is the choice of parameter σ_p. The update rule can be derived using the gradient descent method. However, we experimentally determined that this could lead to instabilities in the learning process. We employ a simpler approach, and anneal σ_p using a specific schedule, $\sigma_p(t+1) = \sigma_p(0)\cdot\sigma_T/(\sigma_T + t)$, where σ_T is the decay parameter. However, we do not wish σ_p to drop to a value near zero as that would result in possibly suboptimal hard prototype assignments, thus we resort to using a validation data set and continue decreasing σ_p as long as $loss_{smp}$ value is being improved on the validation set. Lastly, σ_y^2 is set

to the mean of squared Frobenius norm $\|\mathbf{Y}_n - \hat{\mathbf{Y}}_n\|_F^2$ of the current model.

3.1 Low-rank preference matrix approximation

Since the preference matrix $\mathbf{Q}_k$ is of size $L \times L$, memory required to store preference matrix even for a single prototype can be prohibitively large when number of labels L is very high. In such cases we can resort to low-rank matrix approximation of preference matrix using singular value decomposition (SVD), by representing a matrix as

$$\mathbf{Q}_k = \mathbf{U}_k \cdot \mathbf{\Sigma}_k \cdot \mathbf{V}_k^{\mathrm{T}}, \tag{8}$$

where $\mathbf{U}_k$ and $\mathbf{V}_k$ are $L \times m$ matrices, while $\mathbf{\Sigma}_k$ is a diagonal $m \times m$ matrix for some $m \leq L$. Low-rank matrix approximation allows us to work with large matrices $\mathbf{Q}_k$ by actually storing $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, and $\mathbf{V}_k$ to memory, and can also help filter out noise from $\mathbf{Q}_k$. Substituting equation (8) to (6), and taking derivatives with respect to matrices $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, and $\mathbf{V}_k$ we can easily obtain update rules for $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, and $\mathbf{V}_k$ to replace the $\mathbf{Q}_k$ updates in (7).

4 Pairwise Preference Aggregation algorithm

For an instance $\mathbf{x}$, the SMP-Rank output is represented by pairwise preference matrix $\mathbf{Y}$. The aggregation algorithm computes the total order given these, often conflicting, pairwise preferences. Since pairwise preferences are not transitive in general, inferring total order of labels is not a trivial task. If a measure of goodness of the total order is defined as

$$AGREE(\pi, \mathbf{Y}) = \sum_{i,j:\pi^{-1}(i)>\pi^{-1}(j)} \mathbf{Y}(i,j), \tag{9}$$

where $\pi^{-1}(i) > \pi^{-1}(j)$ if and only if label y_i is preferred over label y_j in total order π, the task to find π_* that maximizes (9) is NP-complete [Cohen *et al.*, 1999]. Several heuristic methods have been proposed to obtain the approximate solution to this problem. One of the first were Borda count [de Borda, 1781], which sorts the labels by the counts of victories over the opponents, and Soft Borda, which sorts by the sums of preference degrees over the opponents. Several other sorting-by-degree methods exist [Cohen *et al.*, 1999; **?**], as well as the approach from [Ailon and Mohri, 2008] which we refer to as QuickSort, as it is based on the popular sorting algorithm [Hoare, 1962]. A recent paper [Ailon, 2012] presents a near-optimal algorithm NearOpt with complexity $\mathcal{O}(\epsilon^{-6} L \log L^6)$ for a regret of ϵ times the optimal loss.

We propose a novel aggregation algorithm, called TOUGH, with the same worst-case theoretical guarantees as [Cohen *et al.*, 1999]. However, it is both simpler and more accurate than current state-of-the-art, as we show in Section 5.1. The algorithm starts with empty ordered list $\hat{\pi}$. At every iteration, it expands the current ordered list by greedily adding a new label to the position in $\hat{\pi}$ that maximizes (9). When the list includes all labels the algorithm terminates. Algorithm 1 shows the TOUGH pseudocode.

Let us represent the preference matrix $\mathbf{Y}$ as a directed graph, where each label is represented by a node, and there is a directed edge between every i-th and j-th node with weight $\mathbf{Y}(i,j)$. Then, the following lemma holds for Algorithm 1.

Algorithm 1 Total-Order-Under-Greedy-Heuristic (TOUGH)

Inputs: label instance set $\Omega = \{1, 2, ..., L\}$, preference matrix $\mathbf{Y}$
Output: approximately optimal ordering $\hat{\pi}$

1. **Initialize** $\hat{\pi}$ to empty list
2. **while** (Ω is not empty)
3. **Find** highest value in $\mathbf{Y}$ and take the preferred label $temp$
4. **Try** every place in $\hat{\pi}$ for $temp$, and put in the place maximizing (9) of $\hat{\pi}$
5. **Set** to 0 entries in $\mathbf{Y}$ between $temp$ and other members of $\hat{\pi}$
6. **Delete** $temp$ from Ω

Lemma 1. *Let $\hat{\pi}_{t-1}$ be the total order after the first $t-1$ iterations of TOUGH, and $\hat{\pi}_t$ be the total order after the t-th iteration (i.e., after addition of $temp$, see Algorithm 1). Furthermore, let a_t and d_t be the sums of all outgoing and incoming edges at the t-th iteration, respectively, from $temp$ to the nodes in $\hat{\pi}_{t-1}$. Then, Δ_t, defined as the contribution of $temp$ at the t-th iteration to the measure of goodness (9) is bounded below as*

$$\Delta_t = AGREE(\hat{\pi}_t, \mathbf{Y}) - AGREE(\hat{\pi}_{t-1}, \mathbf{Y}) \geq \frac{1}{2} \cdot (a_t + d_t).$$

Proof. Consider putting $temp$ at the first place of the current total order list, and then at the last place, which completely reverses the chosen edges. Δ_t for one of these two positions must be larger than or equal to one half of the sum of all the weights from $temp$ to nodes in current total order, since sum of Δ_t of these two positions is exactly equal to the sum of all the weights from $temp$ to nodes in a current total order. □

The lemma states that, when searching for the position for $temp$ (Algorithm 1, step 4), there is always a position such that the sum of the $temp$'s weights that will be included in the new measure of goodness is larger than or equal to one half of the sum of all weights of edges that connect $temp$ to nodes in the current total order. Using Lemma 1, it can be shown that the approximate solution $\hat{\pi}$ is within a factor of 2 to the optimal solution π_*.

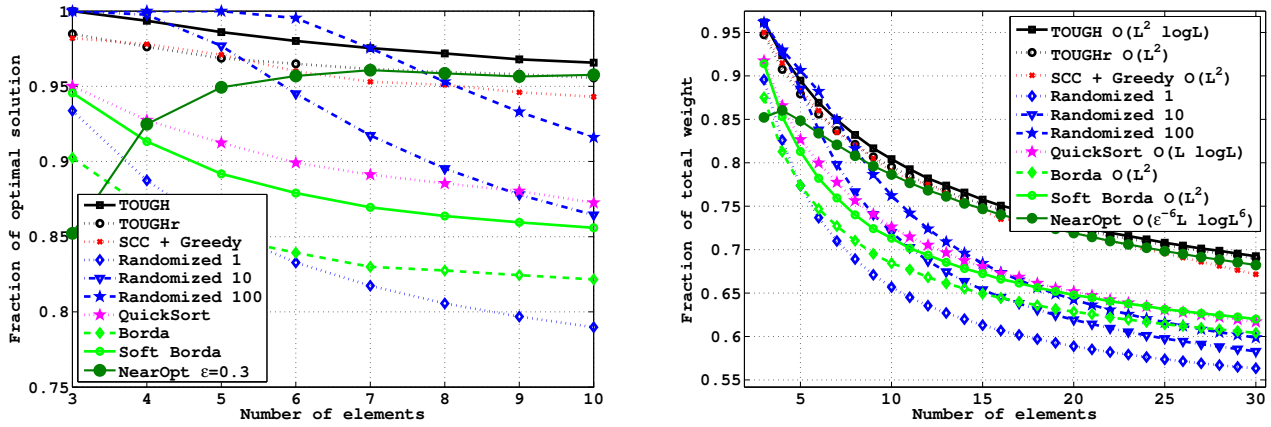
Theorem 1. *Let $OPT(\mathbf{Y})$ be the weighted agreement (9) achieved by an optimal total order π_* for the preference matrix $\mathbf{Y}$, and let $APPROX(\mathbf{Y})$ be the weighted agreement achieved by output of TOUGH algorithm $\hat{\pi}$. Then,*

$$APPROX(\mathbf{Y}) \geq \frac{1}{2} OPT(\mathbf{Y}). \tag{10}$$

Proof. It is clear that $APPROX(\mathbf{Y}) = \sum_{t=1}^{L} \Delta_t$, where Δ_t is defined as in Lemma 1, and L is the number of labels (nodes). Then, using the result of Lemma 1, it follows that $APPROX(\mathbf{Y}) = \sum_{t=1}^{L} \Delta_t \geq \frac{1}{2} \sum_{t=1}^{L} (a_t + d_t)$. Further, following [Cohen *et al.*, 1999], we have

$$\begin{aligned} OPT(\mathbf{Y}) &\leq \sum_{t=1}^{L} (a_t + d_t) = \frac{2}{2} \sum_{t=1}^{L} (a_t + d_t) \\ &\leq 2 \cdot APPROX(\mathbf{Y}). \end{aligned} \tag{11}$$

The first inequality holds because $OPT(\mathbf{Y})$ can at best include every edge in the graph, and since every edge is removed exactly once (weight of edge set to 0), each edge must contribute to some a_t or some d_t. □



(a) Smaller label set

(b) Larger label set

Figure 1: Comparison of pairwise preference aggregation algorithms

The time complexity of TOUGH is $\mathcal{O}(L^2 \log L)$. A simple modification in Step 3 of Algorithm 1 that takes a random label instead of the highest preferred label results in a TOUGHr algorithm with $\mathcal{O}(L^2)$ time complexity. Theorem 10 holds for TOUGHr as well and, as the results show, its performance converges to TOUGH performance for large L.

A popular aggregation algorithm is SCC+Greedy [Cohen *et al.*, 1999], which first orders strongly connected components (SCC) and then the nodes within each SCC, restricting a node to stay in a part of the rank determined by its SCC. On the other hand, TOUGH tries out all possible locations in the partial ordering, which produces a more consistent total ordering. TOUGH is straightforward to implement, e.g., much simpler than SCC+Greedy, which requires a preprocessing step that finds SCC in a graph.

Finally, we note that the authors of the QuickSort algorithm [Ailon and Mohri, 2008] prove regret bound and loss bound seemingly better than the one we show. However, as their algorithm is non-deterministic, the loss bound is on the expectation of loss, where expectation is over random choices of pivot labels. In practice, it achieves bad results if pivot is chosen poorly, performing worse than the bound we give. Therefore, QuickSort does not have this worst-case guarantee, unlike SCC+Greedy and TOUGH algorithms.

5 Empirical Evaluation

5.1 Preference Aggregation Evaluation

We compared 8 different aggregation algorithms: SCC+Greedy, Borda count, Soft Borda, QuickSort, NearOpt, TOUGH, TOUGHr and Randomized. Randomized algorithm is a heuristic that chooses the better between some random order and its inverse ($L, 10L$ and $100L$ random orders were compared for each matrix). We randomly generated 10,000 $\mathbf{Y}$ matrices of sizes ranging from 3 to 30, where element $\mathbf{Y}(i,j)$ is sampled uniformly from $[0,1]$ and $\mathbf{Y}(j,i) = 1 - \mathbf{Y}(i,j)$. For pairwise preference matrices of size up to and including 10, we calculated the optimal total ordering π_* using exhaustive search. Given the approximate total order $\hat{\pi}$ we calculated the fraction of optimal solution as

$$\frac{\sum_{i,j:\hat{\pi}^{-1}(i)>\hat{\pi}^{-1}(j)} \max\{\mathbf{Y}(i,j) - \mathbf{Y}(j,i), 0\}}{\sum_{i,j:\pi_*^{-1}(i)>\pi_*^{-1}(j)} \max\{\mathbf{Y}(i,j) - \mathbf{Y}(j,i), 0\}}. \quad (12)$$

For matrices of size over 10 we did not calculate the optimal total order due to high computational cost. Instead, given $\hat{\pi}$ we calculated the fraction of total weight as

$$\frac{\sum_{i,j:\hat{\pi}^{-1}(i)>\hat{\pi}^{-1}(j)} \max\{\mathbf{Y}(i,j) - \mathbf{Y}(j,i), 0\}}{\sum_{i,j} \max\{\mathbf{Y}(i,j) - \mathbf{Y}(j,i), 0\}}. \quad (13)$$

The results of the comparison are given in Figures 1a and 1b. Despite its simplicity, TOUGH outperformed all other aggregation methods, except for $L \leq 6$, where Randomized$_{100}$ is better because it performs exhaustive search. Note that Randomized$_{10}$ approximates the number of operations in TOUGH and is more expensive than TOUGHr.

At the first glance, Figure 1 seems to show that the performance improvement of TOUGH is not large. However, by taking a closer look, we can see that for $L = 10$, fraction of the optimal solution of TOUGH is 3% smaller than the optimal brute force algorithm. SCC+Greedy is 6% worse than the optimal ($2\times$ performance decrease when compared to TOUGH), while the other algorithms such as QuickSort and Borda are over 10% worse than the optimal solution.

Finally, NearOpt falls well behind TOUGH for $\epsilon = 0.3$. We have experimented with values of $\epsilon = 0.1$ and 0.2 as well. Our conclusion was that it exceeded reasonable time requirements for bigger L, while achieving same or slightly better performance than TOUGH. For example, the $\epsilon = 0.1$ experiment for $L = 10$ could not finish in 1 day.

5.2 Evaluation of Label Ranking Techniques

Here we evaluate the performance of SMP-Rank on label ranking problems. Our preliminary results showed that SMP-

Table 1: Label ranking performance comparison in terms of label rank loss $loss_{LR}$ (**m. p.** - missing preferences)

	complete ranking						30% **missing labels**						60% **missing labels**						30% **m. p.**	
Data Set	IB $_{PL}$	IB $_{Mal}$	Lin $_{LL}$	Lin $_{PL}$	PW TGH	SMP TGH	IB $_{PL}$	IB $_{Mal}$	Lin $_{LL}$	Lin $_{PL}$	PW TGH	SMP TGH	IB $_{PL}$	IB $_{Mal}$	Lin $_{LL}$	Lin $_{PL}$	PW TGH	SMP TGH	PW TGH	SMP TGH
fried	.202	.196	.344	.347	.180	**.144**	.216	.219	.345	.348	.189	**.171**	.257	.312	.349	.350	.192	**.180**	.188	**.175**
calhous.	.337	**.336**	.449	.435	.348	.346	**.345**	.348	.450	.436	.350	.347	.376	.415	.453	.437	.366	**.349**	.349	**.345**
elevators	.234	.226	.306	.305	**.186**	**.186**	.238	.244	.309	.306	.221	**.193**	.263	.322	.311	.307	.257	**.204**	.219	**.194**
pendigits	.143	.139	.364	.351	**.134**	.152	.152	.159	.366	.354	**.143**	.166	.169	.230	.368	.355	**.153**	.172	**.142**	.165
cpu	.344	**.336**	.378	.361	.337	**.336**	.351	.350	.379	.369	.348	**.337**	.371	.408	.380	.370	.373	**.342**	.345	**.337**
segment	.089	.089	.241	.227	.094	**.070**	.111	.117	.242	.228	.099	**.084**	.124	.154	.242	.226	.121	**.093**	.099	**.097**
stock	.106	.108	.283	.268	.163	**.091**	.134	.133	.284	.269	.163	**.113**	.153	.169	.285	.272	.164	**.142**	.171	**.122**
vehicle	.082	.079	.126	.115	.067	**.064**	.093	.095	.129	.118	.077	**.071**	.113	.131	.128	.125	.091	**.082**	.076	**.071**
authorsh.	.063	.062	.182	.175	.082	**.061**	.068	.072	.189	.178	.086	**.065**	.085	.108	.200	.190	.092	**.085**	.109	**.058**
vowel	.158	.148	.313	.317	.140	**.112**	.217	.178	.316	.318	.167	**.151**	.234	.240	.329	.327	.225	**.191**	.179	**.168**
housing	.329	.334	.398	.385	.334	**.305**	.355	.357	.404	.386	.345	**.328**	.384	.385	.408	.391	.366	**.350**	.343	**.335**
bodyfat	.442	.438	.422	.404	.417	**.409**	.463	.448	.430	**.406**	.419	.436	.466	.460	.431	.429	**.426**	.440	.423	**.418**
glass	.098	.094	.107	.106	.080	**.076**	.107	.111	.111	.108	**.085**	**.085**	.115	.140	.124	.118	.101	**.098**	.107	**.100**
wisconsin	.435	.419	.403	**.401**	.418	.408	.434	.427	.409	**.403**	.438	.431	.445	.443	.422	**.412**	.441	.444	.433	**.400**
wine	.040	.036	.067	.059	.038	**.034**	.041	.042	.071	.064	.046	**.040**	.061	.077	.101	.079	.050	**.041**	.046	**.042**
iris	.040	.037	.189	.171	.089	**.029**	.059	.066	.199	.172	.100	**.037**	.103	.118	.208	.174	.105	**.080**	.111	**.048**
sushi	.348	.349	.459	.462	.335	**.332**	.363	.352	.461	.465	.344	**.335**	.381	.374	.463	.468	.366	**.342**	.339	**.333**
cold	.386	.387	.409	.399	**.370**	.384	.398	.398	.410	.404	**.373**	.380	.424	.423	.411	**.405**	.406	.405	.372	**.369**
diau	.336	.332	.350	.349	**.328**	.337	.338	.340	.353	.350	**.331**	.338	.339	.365	.360	.352	**.335**	.353	.322	**.320**
dtt	.418	.417	.421	.411	**.399**	.409	.428	.430	.422	.417	**.404**	.421	.447	.442	.423	.419	.411	.422	**.404**	.407
heat	.438	.435	.439	.438	.435	**.432**	.445	**.433**	.441	.442	.434	**.433**	.460	.455	.445	.443	**.439**	.452	.428	**.419**
spo	.438	.433	.437	.443	.431	**.430**	.439	.444	.439	.445	.435	**.433**	.441	.455	.444	.453	.441	**.440**	.424	**.419**
average	.250	.247	.322	.315	.246	**.234**	.263	.262	.325	.318	.255	**.245**	.282	.301	.331	.323	.270	**.258**	.256	**.243**

Rank with TOUGH performed better than with other aggregation algorithms. Therefore, only these results are shown.

We used 22 data sets (Table 1); 17 semi-synthetic data sets obtained by converting benchmark multi-class (A) and regression data (B) from the UCI and Statlog repositories into label ranking data, using the Naïve Bayes (A) and feature-to-label technique (B) [Cheng *et al.*, 2009], and 6 real-world data sets from the medical [Hüllermeier *et al.*, 2008] and food [Kamishima and Akaho, 2006] domains.

SMP-Rank parameter $\alpha_0 = 0.04$ and update step $\alpha(t) = \alpha_0 \alpha_T / (\alpha_T + t)$ were used, where t denotes the training iteration and $\alpha_T = 5N$, with N being the training set size. The parameter σ_p was initialized to the value of within-data variance, and updated by schedule described in Section 3 with $\sigma_T = 5N$. Validation was done using 10% of the training data. Number of prototypes K was set to 100.

SMP-Rank was compared to simple 1-NN rule, instance-based Mallows (IB-Mal) and Plackett-Luce (IB-PL) models, two log-linear label ranking models (Lin-LL and Lin-PL), and to the PW approach [Hüllermeier *et al.*, 2008] that uses RBF kernel SVM classifiers as base models. Preliminary results showed that using TOUGH aggregation with PW instead of originally proposed Soft-Borda greatly improves performance, therefore TOUGH was used with PW. For IB algorithms, the neighborhood size $k \in \{5, 10, 15, 20\}$ was selected through cross-validation.

Table 1 shows the results obtained using training data with complete rankings, 30% and 60% of missing labels (partial orders), and 30% missing preferences. The results are given in terms of average $loss_{LR}$ (1) after five repetitions of a ten-fold cross-validation. Partial orders were simulated by removing uniformly at random a certain percentage of labels from the training data. Missing preferences were simulated by removing uniformly at random a percentage of pairwise preferences. Since only SMP-Rank and PW can learn from partial preferences that do not form an order, only these two algorithms were compared in this setting. To account for missing preferences between labels y_i and y_j, SMP-Rank simply ignored fields $\mathbf{Y}(i, j)$ and $\mathbf{Y}(j, i)$ during training.

It can be observed that SMP-Rank$_{tough}$ achieved the best overall accuracy and consistently outperformed the competitors, particularly on large data sets and when confronted with the missing label problem. For example, it outperforms the best considered algorithm (PW$_{tough}$) 16 out of 22 times for 60% missing labels, while the worst-performing algorithm (Lin-LL) is outperformed 20 out of 22 times.

The dominance over the IB-based algorithms can be explained by the fact that the performance of IB algorithms is highly dependent on the quality of the training data, since no abstraction is made during the training phase. Clearly, it is advantageous to use a model that abstracts over more data, thus alleviating influence of noise. If we try to achieve this by considering a larger number of neighbors in the IB algorithms, we start ignoring distances between an instance and its neighbors as a similarity measure. On the other hand, SMP-Rank generalizes training data to produce a representation in terms of prototype vectors, effectively utilizing distances to prototypes as a similarity measure.

6 Conclusion

We introduced SMP-Rank algorithm for prediction of label rankings with a novel pairwise preference aggregation algorithm for which we prove the worst-case guarantee. The algorithm is capable of operating in an online manner, it is memory-efficient since it operates on predefined budget, and it carries out pairwise preference aggregation more accurately than the previously proposed methods. Empirical investigation indicates that SMP-Rank produces better results than the current state-of-the-art, especially when learning from incomplete label preferences where the performance is improved by the largest margin.

References

[Ailon and Mohri, 2008] Nir Ailon and Mehryar Mohri. An Efficient Reduction of Ranking to Classification. In *COLT*, pages 87–98. Omnipress, 2008.

[Ailon, 2012] Nir Ailon. An Active Learning Algorithm for Ranking from Pairwise Preferences. *Journal of Machine Learning Research*, 13(137–164), 2012.

[Balcan *et al.*, 2007] Maria-Florina Balcan, Nikhil Bansal, Alina Beygelzimer, Don Coppersmith, John Langford, and Gregory B. Sorkin. Robust Reductions from Ranking to Classification. In *COLT*, volume 4539 of *Lecture Notes in Computer Science*, pages 604–619, 2007.

[Boley *et al.*, 1999] Daniel Boley, Maria Gini, Robert Gross, Eui-Hong (Sam) Han, Kyle Hastings, George Karypis, Vipin Kumar, Bamshad Mobasher, and Jerome Moore. Partitioning-based clustering for Web document categorization. *Decision Support Systems*, 27(3):329–341, 1999.

[Cao *et al.*, 2007] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[Cheng *et al.*, 2009] Weiwei Cheng, Jens Hühn, and Eyke Hüllermeier. Decision tree and instance-based learning for label ranking. In *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, pages 161–168, 2009.

[Cheng *et al.*, 2010] Weiwei Cheng, Krzysztof Dembczyński, and Eyke Hüllermeier. Label ranking methods based on the Plackett-Luce model. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 215–222, 2010.

[Cohen *et al.*, 1999] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[de Borda, 1781] Jean Charles de Borda. *Memoire sur les Élections au Scrutin*. Histoire de l'Academie Royale des Sciences, 1781.

[Dekel *et al.*, 2003] Ofer Dekel, Christopher Manning, and Yoram Singer. Log-Linear Models for Label Ranking. In *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2003.

[Gärtner and Vembu, 2010] Thomas Gärtner and Shankar Vembu. Label Ranking Algorithms: A Survey. In Eyke Hüllermeier Johannes Fürnkranz, editor, *Preference Learning*. Springer–Verlag, 2010.

[Grbovic and Vucetic, 2009] Mihajlo Grbovic and Slobodan Vucetic. Regression learning vector quantization. *IEEE International Conference on Data Mining*, pages 788–793, 2009.

[Grbovic *et al.*, 2012] Mihajlo Grbovic, Nemanja Djuric, and Slobodan Vucetic. Supervised clustering of label ranking data. *SIAM International Conference on Data Mining*, 2012.

[Har-Peled *et al.*, 2003] Sariel Har-Peled, Dan Roth, and Dav Zimak. Constraint classification for multiclass classification and ranking. In *Proceedings of the 16th Annual Conference on Neural Information Processing Systems, NIPS-02*, pages 785–792. MIT Press, 2003.

[Hart, 1968] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, IT-14:515–516, 1968.

[Hoare, 1962] Charles Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[Hüllermeier *et al.*, 2008] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172:1897–1916, 2008.

[Kamishima and Akaho, 2006] Toshihiro Kamishima and Shotaro Akaho. Efficient Clustering for Orders. In *ICDM Workshops*, pages 274–278. IEEE Computer Society, 2006.

[Kohonen, 1990] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78:1464–1480, 1990.

[Lu and Boutilier, 2011] Tyler Lu and Craig Boutilier. Learning Mallows Models with Pairwise Preferences. *International Conference on Machine Learning*, 2011.

[Mallows, 1967] C. L. Mallows. Non-null ranking models. *Biometrika*, 44:114–130, 1967.

[Park and Chu, 2009] Seung-Taek Park and Wei Chu. Pairwise preference regression for cold-start recommendation. *Proceedings of the third ACM conference on Recommender systems*, 2009.

[Plackett, 1975] R. L. Plackett. The analysis of permutations. *Applied Statistics*, 24(2):193–202, 1975.

[Qin *et al.*, 2010] Tao Qin, Xiubo Geng, and Tie-Yan Liu. A New Probabilistic Model for Rank Aggregation. In *Advances in Neural Information Processing Systems 23*, pages 1948–1956. MIT Press, 2010.

[Seo and Obermayer, 2003] Sambu Seo and Klaus Obermayer. Soft learning vector quantization. *Neural computation*, 15(7):1589–1604, 2003.

[Weigend *et al.*, 1995] Andreas S. Weigend, Morgan Mangeas, and Ashok N. Srivastava. Nonlinear Gated Experts for Time Series: Discovering Regimes and Avoiding Overfitting. *International Journal of Neural Systems*, 6:373–399, 1995.

[Zhou *et al.*, 2008] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. *Algorithmic Aspects in Information and Management*, pages 337–348, 2008.