SQL-Rank: A Listwise Approach to Collaborative Ranking

Liwei Wu [1 2] **Cho-Jui Hsieh** [1 2] **James Sharpnack** [1]

Abstract

In this paper, we propose a listwise approach for constructing user-specific rankings in recommendation systems in a collaborative fashion. We contrast the listwise approach to previous pointwise and pairwise approaches, which are based on treating either each rating or each pairwise comparison as an independent instance respectively. By extending the work of (Cao et al., 2007), we cast listwise collaborative ranking as maximum likelihood under a permutation model which applies probability mass to permutations based on a low rank latent score matrix. We present a novel algorithm called SQL-Rank, which can accommodate ties and missing data and can run in linear time. We develop a theoretical framework for analyzing listwise ranking methods based on a novel representation theory for the permutation model. Applying this framework to collaborative ranking, we derive asymptotic statistical rates as the number of users and items grow together. We conclude by demonstrating that our SQL-Rank method often outperforms current state-of-the-art algorithms for implicit feedback such as Weighted-MF and BPR and achieve favorable results when compared to explicit feedback algorithms such as matrix factorization and collaborative ranking.

1. Introduction

We study a novel approach to collaborative ranking—the personalized ranking of items for users based on their observed preferences—through the use of listwise losses, which are dependent only on the observed rankings of items by users. We propose the SQL-Rank algorithm, which can handle ties and missingness, incorporate both explicit ratings and more implicit feedback, provides personalized rankings, and is based on the relative rankings of items. To better understand the proposed contributions, let us begin with a brief history of the topic.

1.1. A brief history of collaborative ranking

Recommendation systems, found in many modern web applications, movie streaming services, and social media, rank new items for users and are judged based on user engagement (implicit feedback) and ratings (explicit feedback) of the recommended items. A high-quality recommendation system must understand the popularity of an item and infer a user's specific preferences with limited data. Collaborative filtering, introduced in (Hill et al., 1995), refers to the use of an entire community's preferences to better predict the preferences of an individual (see (Schafer et al., 2007) for an overview). In systems where users provide ratings of items, collaborative filtering can be approached as a pointwise prediction task, in which we attempt to predict the unobserved ratings (Pan et al., 2017). Low rank methods, in which the rating distribution is parametrized by a low rank matrix (meaning that there are a few latent factors) provides a powerful framework for estimating ratings (Mnih & Salakhutdinov, 2008; Koren, 2008). There are several issues with this approach. One issue is that the feedback may not be representative of the unobserved entries due to a sampling bias, an effect that is prevalent when the items are only 'liked' or the feedback is implicit because it is inferred from user engagement. Augmenting techniques like weighting were introduced to the matrix factorization objective to overcome this problem (Hsieh et al., 2015; Hu et al., 2008). Many other techniques are also introduced (Kabbur et al., 2013; Wang et al., 2017; Wu et al., 2016). Another methodology worth noting is the CofiRank algorithm of (Weimer et al., 2008) which minimizes a convex surrogate of the normalized discounted cumulative gain (NDCG). The pointwise framework has other flaws, chief among them is that in recommendation systems we are not interested in predicting ratings or engagement, but rather we must rank the items.

Ranking is an inherently relative exercise. Because users have different standards for ratings, it is often desirable for ranking algorithms to rely only on relative rankings and not absolute ratings. A ranking loss is one that only considers a user's relative preferences between items, and ignores the

[1]Department of Statistics, University of California, Davis, CA, USA [2]Department of Computer Science, University of California, Davis, CA, USA. Correspondence to: Liwei Wu <liwu@ucdavis.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

absolute value of the ratings entirely, thus deviating from the pointwise framework. Ranking losses can be characterized as pairwise and listwise. A pairwise method decomposes the objective into pairs of items j, k for a user i, and effectively asks 'did we successfully predict the comparison between j and k for user i?'. The comparison is a binary response—user i liked j more than or less than k—with possible missing values in the event of ties or unobserved preferences. Because the pairwise model has cast the problem in the classification framework, then tools like support vector machines were used to learn rankings; (Joachims, 2002) introduces rankSVM and efficient solvers can be found in (Chapelle & Keerthi, 2010). Much of the existing literature focuses on learning a single ranking for all users, which we will call simple ranking (Freund et al., 2003; Agarwal, 2006; Pahikkala et al., 2009). This work will focus on the personalized ranking setting, in which the ranking is dependent on the user.

Pairwise methods for personalized ranking have seen great advances in recent years, with the AltSVM algorithm of (Park et al., 2015), Bayesian personalized ranking (BPR) of (Rendle et al., 2009), and the near linear-time algorithm of (Wu et al., 2017). Nevertheless, pairwise algorithms implicitly assume that the item comparisons are independent, because the objective can be decomposed where each comparison has equal weight. Listwise losses instead assign a loss, via a generative model, to the entire observed ranking, which can be thought of as a permutation of the m items, instead of each comparison independently. The listwise permutation model, introduced in (Cao et al., 2007), can be thought of as a weighted urn model, where items correspond to balls in an urn and they are sequentially plucked from the urn with probability proportional to $\phi(X_{ij})$ where X_{ij} is the latent score for user i and item j and ϕ is some non-negative function. They proposed to learn rankings by optimizing a cross entropy between the probability of k items being at the top of the ranking and the observed ranking, which they combine with a neural network, resulting in the ListNet algorithm. (Shi et al., 2010) applies this idea to collaborative ranking, but uses only the top-1 probability because of the computational complexity of using top-k in this setting. This was extended in (Huang et al., 2015) to incorporate neighborhood information. (Xia et al., 2008) instead proposes a maximum likelihood framework that uses the permutation probability directly, which enjoyed some empirical success.

Very little is understood about the theoretical performance of listwise methods. (Cao et al., 2007) demonstrates that the listwise loss has some basic desirable properties such as monotonicity, i.e. increasing the score of an item will tend to make it more highly ranked. (Lan et al., 2009) studies the generalizability of several listwise losses, using the local Rademacher complexity, and found that the excess risk could be bounded by a $1/\sqrt{n}$ term (recall, n is the number of users). Two main issues with this work are that no dependence on the number of items is given—it seems these results do not hold when m is increasing—and the scores are not personalized to specific users, meaning that they assume that each user is an independent and identically distributed observation. A simple open problem is: can we consistently learn preferences from a single user's data if we are given item features and we assume a simple parametric model? ($n = 1, m \to \infty$.)

1.2. Contributions of this work

We can summarize the shortcomings of the existing work: current listwise methods for collaborative ranking rely on the top-1 loss, algorithms involving the full permutation probability are computationally expensive, little is known about the theoretical performance of listwise methods, and few frameworks are flexible enough to handle explicit and implicit data with ties and missingness. This paper addresses each of these in turn by proposing and analyzing the SQL-rank algorithm.

- We propose the SQL-Rank method, which is motivated by the permutation probability, and has advantages over the previous listwise method using cross entropy loss.
- We provide an $O(\text{iter} \cdot (|\Omega|r))$ linear algorithm based on stochastic gradient descent, where Ω is the set of observed ratings and r is the rank.
- The methodology can incorporate both implicit and explicit feedback, and can gracefully handle ties and missing data.
- We provide a theoretical framework for analyzing listwise methods, and apply this to the simple ranking and personalized ranking settings, highlighting the dependence on the number of users and items.

2. Methodology

2.1. Permutation probability

The permutation probability, (Cao et al., 2007), is a generative model for the ranking parametrized by latent scores. First assume there exists a ranking function that assigns scores to all the items. Let's say we have m items, then the scores assigned can be represented as a vector $s = (s_1, s_2, ..., s_m)$. Denote a particular permutation (or ordering) of the m items as π, which is a random variable and takes values from the set of all possible permutations S_m (the symmetric group on m elements). π_1 denotes the index of highest ranked item and π_m is the lowest ranked. The

probability of obtaining π is defined to be

$$P_s(\pi) := \prod_{j=1}^{m} \frac{\phi(s_{\pi_j})}{\sum_{l=j}^{m} \phi(s_{\pi_l})}, \tag{1}$$

where $\phi(.)$ is an increasing and strictly positive function. An interpretation of this model is that each item is drawn without replacement with probability proportional to $\phi(s_i)$ for item i in each step. One can easily show that $P_s(\pi)$ is a valid probability distribution, i.e. $\sum_{\pi \in S_m} P_s(\pi) = 1, P_s(\pi) > 0, \forall \pi$. Furthermore, this definition of permutation probability enjoys several favorable properties (see (Cao et al., 2007)). For any permutation π if you swap two elements ranked at $i < j$ generating the permutation π' ($\pi'_i = \pi_j$, $\pi'_j = \pi_i$, $\pi_k = \pi'_k, k \neq i, j$), if $s_{\pi_i} > s_{\pi_j}$ then $P_s(\pi) > P_s(\pi')$. Also, if permutation π satisfies $s_{\pi_i} > s_{\pi_{i+1}}, \forall i$, then we have $\pi = \arg\max_{\pi' \in S_m} P_s(\pi')$. Both of these properties can be summarized: larger scores will tend to be ranked more highly than lower scores. These properties are required for the negative log-likelihood to be considered sound for ranking (Xia et al., 2008).

In recommendation systems, the top ranked items can be more impactful for the performance. In order to focus on the top k ranked items, we can compute the partial-ranking marginal probability,

$$P_s^{(k,\bar{m})}(\pi) = \prod_{j=1}^{\min\{k,\bar{m}\}} \frac{\phi(s_{\pi_j})}{\sum_{l=j}^{\bar{m}} \phi(s_{\pi_l})}. \tag{2}$$

It is a common occurrence that only a proportion of the m items are ranked, and in that case we will allow $\bar{m} \leq m$ to be the number of observed rankings (we assume that $\pi_1, \ldots, \pi_{\bar{m}}$ are the complete list of ranked items). When $k = 1$, the first summation vanishes and top-1 probability can be calculated straightforwardly, which is why $k = 1$ is widely used in previous listwise approaches for collaborative ranking. Counter-intuitively, we demonstrate that using a larger k tends to improve the ranking performance.

We see that computing the likelihood loss is linear in the number of ranked items, which is in contrast to the cross-entropy loss used in (Cao et al., 2007), which takes exponential time in k. The cross-entropy loss is also not sound, i.e. it can rank worse scoring permutations more highly, but the negative log-likelihood is sound. We will discuss how we can deal with ties in the following subsection, namely, when the ranking is derived from ratings and multiple items receive the same rating, then there is ambiguity as to the order of the tied items. This is a common occurrence when the data is implicit, namely the output is whether the user engaged with the item or not, yet did not provide explicit feedback. Because the output is binary, the cross-entropy loss (which is based on top-k probability with k very small) will perform very poorly because there will be many ties for the top ranked items. To this end, we propose a collaborative ranking algorithm using the listwise likelihood that can accommodate ties and missingness, which we call Stochastic Queuing Listwise Ranking, or SQL-Rank.

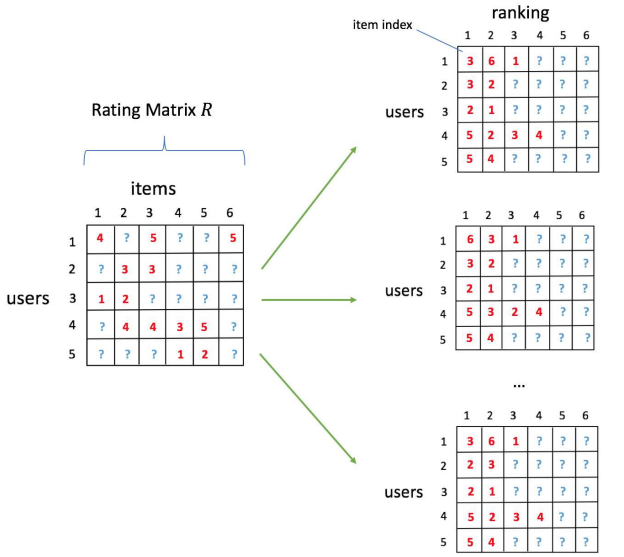

Figure 1. Demonstration of Stochastic Queuing Process—the rating matrix R (left) generates multiple possible rankings Π's (right), $\Pi \in \mathcal{S}(R, \Omega)$ by breaking ties randomly.

2.2. Deriving objective function for SQL-Rank

The goal of collaborative ranking is to predict a personalized score X_{ij} that reflects the preference level of user i towards item j, where $1 \leq i \leq n$ and $1 \leq j \leq m$. It is reasonable to assume the matrix $X \in \mathbb{R}^{n \times m}$ to be low rank because there are only a small number of latent factors contributing to users' preferences. The input data is given in the form of "user i gives item j a relevance score R_{ij}". Note that for simplicity we assume all the users have the same number $\bar{m}$ of ratings, but this can be easily generalized to the non-uniform case by replacing $\bar{m}$ with m_i (number of ratings for user i).

With our scores X and our ratings R, we can specify our collaborative ranking model using the permutation probability (2). Let Π_i be a ranking permutation of items for user i (extracted from R), we can stack $\Pi_1, \ldots \Pi_n$, row by row, to get the permutation matrix $\Pi \in \mathbb{R}^{n \times m}$. Assuming users are independent with each other, the probability of observing a particular Π given the scoring matrix X can be written as

$$P_X^{(k,\bar{m})}(\Pi) = \prod_{i=1}^{n} P_{X_i}^{(k,\bar{m})}(\Pi_i). \tag{3}$$

We will assume that $\log \phi(x) = 1/(1 + \exp(-x))$ is the sigmoid function. This has the advantage of bounding the resulting weights, $\phi(X_{ij})$, and maintaining their positivity without adding additional constraints.

Typical rating data will contain many *ties* within each row. In such cases, the permutation Π is no longer unique and there is a set of permutations that coincides with rating because with any candidate Π we can arbitrarily shuffle the ordering of items with the same relevance scores to generate a new candidate matrix Π' which is still valid (see Figure 1). We denote the set of valid permutations as $\mathcal{S}(R, \Omega)$, where Ω is the set of all pairs (i, j) such that $R_{i,j}$ is observed. We call this shuffling process the *Stochastic Queuing Process*, since one can imagine that by permuting ties we are stochastically queuing new Π's for future use in the algorithm.

The probability of observing R therefore should be defined as $P_X^{(k,\bar{m})}(R) = \sum_{\Pi \in \mathcal{S}(R,\Omega)} P_X(\Pi)$. To learn the scoring matrix X, we can naturally solve the following maximum likelihood estimator with low-rank constraint:

$$\min_{X \in \mathcal{X}} -\log \sum_{\Pi \in \mathcal{S}(R,\Omega)} P_X^{(k,\bar{m})}(\Pi), \tag{4}$$

where $\mathcal{X}$ is the structural constraint of the scoring matrix. To enforce low-rankness, we use the nuclear norm regularization $\mathcal{X} = \{X : \|X\|_* \leq r\}$.

Eq (4) is hard to optimize since there is a summation inside the $\log$. But by Jensen's inequality and convexity of $-\log$ function, we can move the summation outside $\log$ and obtain an upper bound of the original negative log-likelihood, leading to the following optimization problem:

$$\min_{X \in \mathcal{X}} -\sum_{\Pi \in \mathcal{S}(R,\Omega)} \log P_X^{(k,\bar{m})}(\Pi) \tag{5}$$

This upper bound is much easier to optimize and can be solved using Stochastic Gradient Descent (SGD).

Next we discuss how to apply our model for explicit and implicit feedback settings. In the explicit feedback setting, it is assumed that the matrix R is partially observed and the observed entries are explicit ratings in a range (e.g., 1 to 5). We will show in the experiments that $k = \bar{m}$ (using the full list) leads to the best results. (Huang et al., 2015) also observed that increasing k is useful for their cross-entropy loss, but they were not able to increase k since their model has time complexity exponential to k.

In the implicit feedback setting each element of R_{ij} is either 1 or 0, where 1 means positive actions (e.g., click or like) and 0 means no action is observed. Directly solving (5) will be expensive since $\bar{m} = m$ and the computation will involve all the mn elements at each iteration. Moreover, the 0's in the matrix could mean either a lower relevance score or missing, thus should contribute less to the objective function. Therefore, we adopt the idea of negative sampling (Mikolov et al., 2013) in our list-wise formulation. For each user (row of R), assume there are $\tilde{m}$ 1's, we then sample $\rho\tilde{m}$ unobserved entries uniformly from the same row and append to the back of the list. This then becomes the problem with $\bar{m} = (1 + \rho)\tilde{m}$ and then we use the same algorithm in explicit feedback setting to conduct updates. We then repeat the sampling process at the end of each iteration, so the update will be based on different set of 0's at each time.

Algorithm 1 SQL-Rank: General Framework

Input: Ω, $\{R_{ij} : (i, j) \in \Omega\}$, $\lambda \in \mathbb{R}^+$, ss, $rate$, ρ
Output: $U \in \mathbb{R}^{r \times n}$ and $V \in \mathbb{R}^{r \times m}$
Randomly initialize U, V from Gaussian Distribution
repeat
  Generate a new permutation matrix Π {see alg 2}
  Apply gradient update to U while fixing V
  Apply gradient update to V while fixing U {see alg 4}
until performance for validation set is good
return U, V {recover score matrix X}

Algorithm 2 Stochastic Queuing Process

Input: Ω, $\{R_{ij} : (i, j) \in \Omega\}$, ρ
Output: $\Pi \in \mathbb{R}^{n \times m}$
for $i = 1$ **to** n **do**
  Sort items based on observed relevance levels R_i
  Form Π_i based on indices of items in the sorted list
  Shuffle Π_i for items within the same relevance level
  if Dataset is implicit feedback **then**
    Uniformly sample $\rho\tilde{m}$ items from unobserved items
    Append sampled indices to the back of Π_i
  end if
end for
Stack Π_i as rows to form matrix Π
Return Π {Used later to compute gradient}

2.3. Non-convex implementation

Despite the advantage of the objective function in equation (5) being convex, it is still not feasible for large-scale problems since the scoring matrix $X \in \mathbb{R}^{n \times m}$ leads to high computational and memory cost. We follow a common trick to transform (5) to the non-convex form by replacing $X = U^T V$: with $U \in \mathbb{R}^{r \times n}, V \in \mathbb{R}^{r \times m}$ so that the objective is,

$$\sum_{\Pi \in \mathcal{S}(R,\Omega)} \underbrace{-\sum_{i=1}^{n} \sum_{j=1}^{\bar{m}} \log \frac{\phi(u_i^T v_{\Pi_{ij}})}{\sum_{l=j}^{\bar{m}} \phi(u_i^T v_{\Pi_{il}})}}_{f(U,V)} + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2),$$

where u_i, v_j are columns of U, V respectively. We apply stochastic gradient descent to solve this problem. At each step, we choose a permutation matrix $\Pi \in \mathcal{S}(R, \Omega)$ using the stochastic queuing process (Algorithm 2) and then update U, V by $\nabla f(U, V)$. For example, the gradient with

respect to V is ($g = \log \phi$ is the sigmoid function),

$$\frac{\partial f}{\partial v_j} = \sum_{i \in \Omega_j} \sum_{t=1}^{\bar{m}} \Bigg\{ -g'(u_i^T v_t) u_i + \frac{\mathbb{1}(\text{rank}_i(j) \geq t) \phi(u_i^T v_j)}{\sum_{l=t}^{\bar{m}} \phi(u_i^T v_{\Pi_{il}})} g'(u_i^T v_j) u_i \Bigg\}$$

where Ω_j denotes the set of users that have rated the item j and $\text{rank}_i(j)$ is a function gives the rank of the item j for that user i. Because g is the sigmoid function, $g' = g \cdot (1 - g)$. The gradient with respect to U can be derived similarly.

As one can see, a naive way to compute the gradient of f requires $O(n\bar{m}^2 r)$ time, which is very slow even for one iteration. However, we show in Algorithm 3 (in the appendix) that there is a smart way to re-arranging the computation so that $\nabla_V f(U, V)$ can be computed in $O(n\bar{m}r)$ time, which makes our SQL-Rank a linear-time algorithm (with the same per-iteration complexity as classical matrix factorization).

3. Theory

Throughout this section, we will establish a theoretical framework for understanding listwise ranking algorithms. We do not consider ties and missing data and reserve this extension of the theory developed here for future work. These tools can be employed to analyze any problem of the constrained form

$$\hat{X} := \arg\min - \log P_X(\Pi) \text{ such that } X \in \mathcal{X}. \quad (6)$$

We will consider two main settings of listwise ranking, the simple ranking setting where for each $X \in \mathcal{X}$,

$$X_{ij} = \beta^\top z_{ij}, \beta \in \mathbb{R}^s, \|\beta\| \leq c_b, \quad (7)$$

where the feature vectors $z_{ij} \in \mathbb{R}^s$ are known, and the personalized setting,

$$X_{ij} = u_i^\top v_j, u_i, v_j \in \mathbb{R}^r, \|U\|_F \leq c_u, \|V\|_F \leq c_v. \quad (8)$$

The simple ranking setting, among other listwise programs was considered in (Lan et al., 2009), and it was determined that the excess risk is bounded by a $1/\sqrt{n}$ term. Critically, these results assumed that the number of items m is bounded, severely limiting their relevance to realistic recommendation systems. It seems that we should be able to learn something about a user's preferences by having them rank more items, yet the existing theory does not reflect this.

The main engine of our theoretical analysis is a generative mechanism for listwise ranking, which demonstrates that the permutation probability model, (1), is also the probability of a row-wise ordering of an exponential ensemble matrix. We demonstrate that the excess risk in the parametric setting scales like $\sqrt{m} \ln m/n$, achieving parametric rates in n and sub-linear excess risk in m when the feature dimension s is fixed. In the personalized setting, (8), we bound the excess risk by $\sqrt{m/n} \ln m$ when the rank r is fixed, which matches comparable results for matrix factorization up to log factors.

3.1. Generative mechanism for listwise ranking

We give an alternative generative mechanism which will prove useful for understanding the listwise ranking objective.

Theorem 1. *Consider a matrix, Y, with independent entries, Y_{ij} that are drawn from an exponential distribution with rate $\phi(X_{ij})$. Let Π_i be the ordering of the entries of Y_i from smallest to largest, then the probability of $\Pi_i | X_i$ is exactly $P_{X_i}(\Pi_i)$.*

The proof is in the appendix. A key aspect of this generative mechanism is that the listwise likelihood can be written as a function of the exponential ensemble. This allows us to establish concentration of measure results for the listwise loss via bounded differences.

3.2. Statistical guarantees

As a first step to controlling the excess risk, we establish a basic inequality. This bounds the excess risk by an empirical process term, which is a random function of $\hat{X}$ and for a fixed $\hat{X}$ it has mean zero. The excess risk (the difference in expected loss between the estimate and the truth) can also be written as the KL divergence between the estimated model and the true model.

Lemma 1. *Consider the minimizer, $\hat{X}$, to the constrained optimization, (6). Suppose that there exists a $X^\star \in \mathcal{X}$ such that $\Pi_i \sim P_{X_i^\star}$ independently for all $i = 1, \ldots, n$. The KL-divergence between the estimate and the truth is bounded*

$$D(X^\star, \hat{X}) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{E} \log \frac{P_{X_i^\star}(\Pi_i)}{P_{\hat{X}_i}(\Pi_i)} \quad \text{(basic)}$$
$$\leq -\frac{1}{n} \sum_{i=1}^{n} \left(\log \frac{P_{X_i^\star}(\Pi_i)}{P_{\hat{X}_i}(\Pi_i)} - \mathbb{E} \log \frac{P_{X_i^\star}(\Pi_i)}{P_{\hat{X}_i}(\Pi_i)} \right).$$

Because the RHS of (basic), the empirical process term, has mean zero and is a function of the random permutation, we can use Theorem 1 to bound it with high probability for a fixed $\hat{X}$. Because $\hat{X}$ is random, we need to control the empirical process term uniformly over the selection of $\hat{X} \in \mathcal{X}$. To this end, we employ Dudley's chaining, which gives us the following theorem (see the Supplement for the complete proof).

Theorem 2. *Assume the conditions of Lemma 1. Define the matrix norm, for the $n \times m$ matrix Z,*

$$\|Z\|_{\infty,2} := \sqrt{\sum_{i=1}^{n} \|Z_i\|_\infty^2}$$

and define $\mathcal{Z} = \{\log \phi(X) : X \in \mathcal{X}\}$ *where* $\log \phi$ *is applied elementwise. Also, let* $\mathcal{N}(\epsilon, \mathcal{Z}, \|.\|_{\infty,2})$ *be the* ϵ*-covering number of* $\mathcal{Z}$ *in the* $\infty, 2$ *norm (the fewest number of* ϵ *radius balls in the* $\infty, 2$ *norm required to cover* $\mathcal{Z}$*). Then, if* $\sup_{Z \in \mathcal{Z}} \|Z\|_\infty \leq C$ *(where* $\|.\|_\infty$ *is the elementwise absolute maximum), then*

$$D(X^\star, \hat{X}) = O_{\mathbb{P}}\left(\frac{\sqrt{m}\ln(m)}{n} \cdot g(\mathcal{Z})\right),$$

where

$$g(\mathcal{Z}) := \int_0^\infty \sqrt{\ln \mathcal{N}(u, \mathcal{Z}, \|.\|_{\infty,2})}\mathrm{d}u,$$

and C *is bounded by a constant in* n, m*.*

Theorem 2 bounds the KL-divergence by the geometric quantity $g(\mathcal{Z})$. For the derived corollaries, we will assume that $\log \phi$ is 1-Lipschitz, which is true when $\log \phi$ is the sigmoid function. The results do not significantly change by increasing the Lipschitz constant, so for simplicity of presentation we set it to be 1.

Corollary 1. *Assume the conditions to Lemma 1, the simple ranking setting (7), that* $\log \phi$ *is 1-Lipschitz, and* $\|Z_{ij}\|_2$ *is bounded uniformly, then*

$$D(X^\star, \hat{X}) = O_{\mathbb{P}}\left(\frac{\sqrt{sm}}{n}\ln m\right).$$

Notably when $n = 1$ this bound is on the order of $\sqrt{m}\ln m$. In the event that $P_{X^\star}$ is concentrated primarily on a single permutation for this user, and we resort to random guessing (i.e. $\hat{X}_{1j} = 0$) then the KL divergence will be close to $\ln m! \approx m \ln m$. So, a reduction of the KL-divergence from order $m \ln m$ to $\sqrt{m}\ln m$ is a large improvement, and the above result should be understood to mean that we can achieve consistency even when $n = 1$ (where consistency is measured relative to random guessing).

Corollary 2. *Assume the conditions to Lemma 1, the personalized ranking setting, (8), and that* $\log \phi$ *is 1-Lipschitz,*

$$D(X^\star, \hat{X}) = O_{\mathbb{P}}\left(\sqrt{\frac{rm}{n}}\ln m\right).$$

Notably, even in the personalized setting, where each user has their own preferences, we can achieve $1/\sqrt{n}$ rates for fixed m, r. Throughout these results the $O_{\mathbb{P}}$ notation only hides the constants c_b, c_u, c_v, and any dependence on s, r, m, n is explicitly given. While Theorem 2 gives us a generic result that can be applied to a large range of constraint sets, we believe that the parametric simple ranking and the low-rank personalized setting are the two most important listwise ranking problems.

4. Experiments

In this section, we compare our proposed algorithm (SQL-Rank) with other state-of-the-art algorithms on real world datasets. Note that our algorithm works for both implicit feedback and explicit feedback settings. In the implicit feedback setting, all the ratings are 0 or 1; in the explicit feedback setting, explicit ratings (e.g., 1 to 5) are given but only to a subset of user-item pairs. Since many real world recommendation systems follow the implicit feedback setting (e.g., purchases, clicks, or checkins), we will first compare SQL-Rank on implicit feedback datasets and show it outperforms state-of-the-art algorithms. Then we will verify that our algorithm also performs well on explicit feedback problems. All experiments are conducted on a server with an Intel Xeon E5-2640 2.40GHz CPU and 64G RAM.

4.1. Implicit Feedback

In the implicit feedback setting we compare the following methods:

- SQL-Rank: our proposed algorithm implemented in Julia [1].
- Weighted-MF: the weighted matrix factorization algorithm by putting different weights on 0 and 1's (Hu et al., 2008; Hsieh et al., 2015).
- BPR: the Bayesian personalized ranking method motivated by MLE (Rendle et al., 2009). For both Weighted-MF and BPR, we use the C++ code by Quora [2].

Note that other collaborative ranking methods such as Pirmal-CR++ (Wu et al., 2017) and List-MF (Shi et al., 2010) do not work for implicit feedback data, and we will compare with them later in the explicit feedback experiments. For the performance metric, we use precision@k for $k = 1, 5, 10$ defined by

$$\text{precision@}k = \frac{\sum_{i=1}^n |\{1 \leq l \leq k : R_{i\Pi_{il}} = 1\}|}{n \cdot k}, \quad (9)$$

where R is the rating matrix and Π_{il} gives the index of the l-th ranked item for user i among all the items not rated by user i in the training set.

We use rank $r = 100$ and tune regularization parameters for all three algorithms using a random sampled validation set. For Weighted-MF, we also tune the confidence weights on unobserved data. For BPR and SQL-Rank, we fix the ratio of subsampled unobserved 0's versus observed 1's to be $3:1$, which gives the best performance for both BPR and SQL-rank in practice.

We experiment on the following four datasets. Note that the

[1] https://github.com/wuliwei9278/SQL-Rank
[2] https://github.com/quora/qmf

Table 1. Comparing implicit feedback methods on various datasets.

DATASET	METHOD	P@1	P@5	P@10
MOVIELENS1M	SQL-RANK	**0.73685**	**0.67167**	0.61833
	WEIGHTED-MF	0.54686	0.49423	0.46123
	BPR	0.69951	0.65608	**0.62494**
AMAZON	SQL-RANK	0.04255	**0.02978**	**0.02158**
	WEIGHTED-MF	0.03647	0.02492	0.01914
	BPR	**0.04863**	0.01762	0.01306
YAHOO MUSIC	SQL-RANK	**0.45512**	**0.36137**	**0.30689**
	WEIGHTED-MF	0.39075	0.31024	0.27008
	BPR	0.37624	0.32184	0.28105
FOURSQUARE	SQL-RANK	**0.05825**	**0.01941**	**0.01699**
	WEIGHTED-MF	0.02184	0.01553	0.01407
	BPR	0.03398	0.01796	0.01359

original data of Movielens1m, Amazon and Yahoo-music are ratings from 1 to 5, so we follow the procedure in (Rendle et al., 2009; Yu et al., 2017) to preprocess the data. We transform ratings of $4, 5$ into 1's and the rest entries (with rating $1, 2, 3$ and unknown) as 0's. Also, we remove users with very few 1's in the corresponding row to make sure there are enough 1's for both training and testing. For Amazon, Yahoo-music and Foursquare, we discard users with less than 20 ratings and randomly select 10 1's as training and use the rest as testing. Movielens1m has more ratings than others, so we keep users with more than 60 ratings, and randomly sample 50 of them as training.

- Movielens1m: a popular movie recommendation data with $6,040$ users and $3,952$ items.
- Amazon: the Amazon purchase rating data for musical instruments [3] with $339,232$ users and $83,047$ items.
- Yahoo-music: the Yahoo music rating data set [4] which contains $15,400$ users and $1,000$ items.
- Foursquare: a location check-in data[5]. The data set contains $3,112$ users and $3,298$ venues with $27,149$ check-ins. The data set is already in the form of "0/1" so we do not need to do any transformation.

The experimental results are shown in Table 1. We find that SQL-Rank outperforms both Weighted-MF and BPR in most cases.

4.2. Explicit Feedback

Next we compare the following methods in the explicit feedback setting:

- SQL-Rank: our proposed algorithm implemented in Julia. Note that in the explicit feedback setting our algorithm only considers pairs with explicit ratings.
- List-MF: the listwise algorithm using the cross entropy loss between observed rating and top 1 probability (Shi et al., 2010). We use the C++ implementation on github[6].
- MF: the classical matrix factorization algorithm in (Koren, 2008) utilizing a pointwise loss solved by SGD. We implemented SGD in Julia.
- Primal-CR++: the recently proposed pairwise algorithm in (Wu et al., 2017). We use the Julia implementation released by the authors[7].

Experiments are conducted on Movielens1m and Yahoo-music datasets. We perform the same procedure as in implicit feedback setting except that we do not need to mask the ratings into "0/1".

We measure the performance in the following two ways:

- NDCG@k: defined as:

$$\text{NDCG@}k = \frac{1}{n}\sum_{i=1}^{n}\frac{\text{DCG@}k(i,\Pi_i)}{\text{DCG@}k(i,\Pi_i^*)},$$

  where i represents i-th user and

$$\text{DCG@}k(i,\Pi_i) = \sum_{l=1}^{k}\frac{2^{R_{i\Pi_{il}}}-1}{log_2(l+1)}.$$

  In the DCG definition, Π_{il} represents the index of the l-th ranked item for user i in test data based on the learned score matrix X. R is the rating matrix and R_{ij} is the rating given to item j by user i. Π_i^* is the ordering provided by the ground truth rating.
- Precision@k: defined as a fraction of relevant items among the top k recommended items:

$$\text{precision@}k = \frac{\sum_{i=1}^{n}|\{1 \le l \le k : 4 \le R_{i\Pi_{il}} \le 5\}|}{n \cdot k},$$

  here we consider items with ratings assigned as 4 or 5 as relevant. R_{ij} follows the same definitions above but unlike before Π_{il} gives the index of the l-th ranked item for user i among all the items that are not rated by user i in the training set (including both rated test items and unobserved items).

As shown in Table 2, our proposed listwise algorithm SQL-Rank outperforms previous listwise method List-MF in both NDCG@10 and precision@$1, 5, 10$. It verifies the claim that log-likelihood loss outperforms the cross entropy loss if we use it correctly. When listwise algorithm SQL-Rank is compared with pairwise algorithm Primal-CR++, the performances between SQL-Rank and Primal-CR++ are quite similar, slightly lower for NDCG@10 but higher for

[3] http://jmcauley.ucsd.edu/data/amazon/

[4] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=3

[5] https://sites.google.com/site/yangdingqi/home/foursquare-dataset

[6] https://github.com/gpoesia/listrankmf

[7] https://github.com/wuliwei9278/ml-1m

Table 2. Comparing explicit feedback methods on various datasets.

DATASET	METHOD	NDCG@10	P@1	P@5	P@10
MOVIELENS1M	SQL-RANK	0.75076	**0.50736**	**0.43692**	**0.40248**
	LIST-MF	0.73307	0.45226	0.40482	0.38958
	PRIMAL-CR++	**0.76826**	0.49365	0.43098	0.39779
	MF	0.74661	0.00050	0.00096	0.00134
YAHOO MUSIC	SQL-RANK	0.66150	**0.14983**	**0.12144**	**0.10192**
	LIST-MF	0.67490	0.12646	0.11301	0.09865
	PRIMAL-CR++	0.66420	0.14291	0.10787	0.09104
	MF	**0.69916**	0.04944	0.03105	0.04787

precision@$1, 5, 10$. Pointwise method MF is doing okay in NDCG but really bad in terms of precision. Despite having comparable NDCG, the predicted top k items given by MF are quite different from those given by other algorithms utilizing a ranking loss. The ordered lists based on SQL-Rank, Primal-CR++ and List-MF, on the other hand, share a lot of similarity and only have minor difference in ranking of some items. It is an interesting phenomenon that we think is worth exploring further in the future.

4.3. Training speed

To illustrate the training speed of our algorithm, we plot precision@1 versus training time for the Movielen1m dataset and the Foursquare dataset. Figure 2 and Figure 3 (in the appendix) show that our algorithm SQL-Rank is faster than BPR and Weighted-MF. Note that our algorithm is implemented in Julia while BPR and Weighted-MF are highly-optimized C++ codes (usually at least 2 times faster than Julia) released by Quora. This speed difference makes sense as our algorithm takes $O(n\bar{m}r)$ time, which is linearly to the observed ratings. In comparison, pair-wise model such as BPR has $O(n\bar{m}^2)$ pairs, so will take $O(n\bar{m}^2 r)$ time for each epoch.

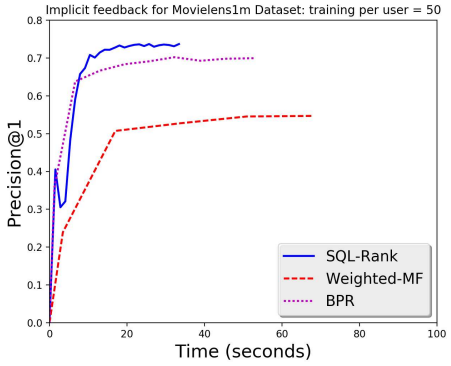

Figure 2. Training time of implicit feedback methods.

4.4. Effectiveness of Stochastic Queuing (SQ)

One important innovation in our SQL-Rank algorithm is the Stochastic Queuing (SQ) Process for handling ties. To illustrate the effectiveness of the SQ process, we compare our algorithm with and without SQ. Recall that without SQ means we fix a certain permutation matrix Π and optimize with respect to it throughout all iterations without generating new Π, while SQ allows us to update using a new permutation at each time. As shown Table 3 and Figure 4 (in the appendix), the performance gain from SQ in terms of precision is substantial (more than 10%) on Movielen1m dataset. It verifies the claim that our way of handling ties and missing data is very effective and improves the ranking results by a lot.

Table 3. Effectiveness of Stochastic Queuing Process.

METHOD	P@1	P@5	P@10
WITH SQ	**0.73685**	**0.67167**	**0.61833**
WITHOUT SQ	0.62763	0.58420	0.55036

4.5. Effectiveness of using the Full List

Another benefit of our algorithm is that we are able to minimize top k probability with much larger k and without much overhead. Previous approaches (Huang et al., 2015) already pointed out increasing k leads to better ranking results, but their complexity is exponential to k so they were not able to have $k > 1$. To show the effectiveness of using permutation probability for full lists rather than using the top k probability for top k partial lists in the likelihood loss, we fix everything else to be the same and only vary k in Equation (5). We obtain the results in Table 4 and Figure 5 (in the appendix). It shows that the larger k we use, the better the results we can get. Therefore, in the final model, we set k to be the maximum number (length of the observed list.)

Table 4. Comparing different k on Movielens1m data set using 50 training data per user.

k	NDCG@10	P@1	P@5	P@10
5	0.64807	0.39156	0.33591	0.29855
10	0.67746	0.43118	0.34220	0.33339
25	0.74589	0.47003	0.42874	0.39796
50 (FULL LIST)	**0.75076**	**0.50736**	**0.43692**	**0.40248**

5. Conclusions

In this paper, we propose a listwise approach for collaborative ranking and provide an efficient algorithm to solve it. Our methodology can incorporate both implicit and explicit feedback, and can gracefully handle ties and missing data. In experiments, we demonstrate our algorithm outperforms existing state-of-the art methods in terms of top k recommendation precision. We also provide a theoretical framework for analyzing listwise methods highlighting the dependence on the number of users and items.

Acknowledgements

JS is partially supported by NSF DMS-1712996. CJH acknowledge the support by NSF IIS-1719097, Google Cloud and Nvidia.

Side Note by Liwei Wu: SQL in SQL-Rank is not only the abbreviation for Stochastically Queuing Listwise, but also name initials of Liwei's girlfriend ShuQing Li. Special thanks for her support.

References

Agarwal, S. Ranking on graph data. In *Proceedings of the 23rd international conference on Machine learning*, pp. 25–32. ACM, 2006.

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

Chapelle, O. and Keerthi, S. S. Efficient algorithms for ranking with svms. *Information Retrieval*, 13(3):201–215, 2010.

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, 4(Nov):933–969, 2003.

Hill, W., Stead, L., Rosenstein, M., and Furnas, G. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 194–201. ACM Press/Addison-Wesley Publishing Co., 1995.

Hsieh, C.-J., Natarajan, N., and Dhillon, I. Pu learning for matrix completion. In *International Conference on Machine Learning*, pp. 2445–2453, 2015.

Hu, Y., Koren, Y., and Volinsky, C. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pp. 263–272. Ieee, 2008.

Huang, S., Wang, S., Liu, T.-Y., Ma, J., Chen, Z., and Veijalainen, J. Listwise collaborative filtering. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 343–352. ACM, 2015.

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

Kabbur, S., Ning, X., and Karypis, G. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 659–667. ACM, 2013.

Koren, Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 426–434. ACM, 2008.

Lan, Y., Liu, T.-Y., Ma, Z., and Li, H. Generalization analysis of listwise learning-to-rank algorithms. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 577–584. ACM, 2009.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Mnih, A. and Salakhutdinov, R. R. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pp. 1257–1264, 2008.

Pahikkala, T., Tsivtsivadze, E., Airola, A., Järvinen, J., and Boberg, J. An efficient algorithm for learning to rank from preference graphs. *Machine Learning*, 75(1):129–165, 2009.

Pan, W., Yang, Q., Duan, Y., Tan, B., and Ming, Z. Transfer learning for behavior ranking. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(5):65, 2017.

Park, D., Neeman, J., Zhang, J., Sanghavi, S., and Dhillon, I. Preference completion: Large-scale collaborative ranking from pairwise comparisons. In *International Conference on Machine Learning*, pp. 1907–1916, 2015.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pp. 452–461. AUAI Press, 2009.

Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. Collaborative filtering recommender systems. In *The adaptive web*, pp. 291–324. Springer, 2007.

Shi, Y., Larson, M., and Hanjalic, A. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pp. 269–272. ACM, 2010.

Talagrand, M. *The generic chaining: upper and lower bounds of stochastic processes*. Springer Science & Business Media, 2006.

Wang, J., Yu, L., Zhang, W., Gong, Y., Xu, Y., Wang, B., Zhang, P., and Zhang, D. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 515–524. ACM, 2017.

Weimer, M., Karatzoglou, A., Le, Q. V., and Smola, A. J. Cofi rank-maximum margin matrix factorization for collaborative ranking. In *Advances in neural information processing systems*, pp. 1593–1600, 2008.

Wu, L., Hsieh, C.-J., and Sharpnack, J. Large-scale collaborative ranking in near-linear time. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 515–524. ACM, 2017.

Wu, Y., DuBois, C., Zheng, A. X., and Ester, M. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pp. 153–162. ACM, 2016.

Xia, F., Liu, T.-Y., Wang, J., Zhang, W., and Li, H. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pp. 1192–1199. ACM, 2008.

Yu, H.-F., Bilenko, M., and Lin, C.-J. Selection of negative samples for one-class matrix factorization. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 363–371. SIAM, 2017.