A Probabilistic Framework for Learning Context-Specific Preferences with Ties*

Truyen Tran[†‡], Dinh Phung[‡] and Svetha Venkatesh[‡]
[†] Department of Computing, Curtin University, WA, Australia
t.tran2@curtin.edu.au
[‡]School of Information Technology, Deakin University, VIC, Australia
{*dinh.phung,svetha.venkatesh*}@*deakin.edu.au*

Abstract

Learning preference models from human generated data is an important task in AI application areas such as decision support systems, information filtering and recommender systems. In many real-world settings, preferences are expressed in the form of simple object ratings indicating the degree to which the objects meet user context-specific needs. Since ratings are often specified within a small numerical range, several objects may have the same ratings, thus creating *ties* among objects for a given context. Dealing with this phenomenon presents a general problem of modelling context-specific *preferences with ties*. To this end, in this paper we describe PMOP - a novel approach by constructing a probabilistic model directly on a finite set of objects and exploiting the combinatorial structure induced by the ties among them. The proposed probabilistic setting allows exploration of a super-exponential combinatorial state space with unknown numbers of partitions and unknown orders. Learning and inference in such a large state space are challenging, and we present in this paper efficient algorithms to perform these tasks. Our approach exploits the discrete choice theory by imposing a generative process such that a finite set of objects is partitioned into subsets in a stagewise procedure, and thus reducing the state-space at each stage significantly. Efficient Markov Chain Monte Carlo (MCMC) algorithms are then derived for the proposed models. We evaluate the models on two application areas: (i) Web document ranking with the data from the recently held Yahoo! challenge, and (ii) movie recommendation in the collaborative filtering setting. We demonstrate that the models are competitive against state-of-the-arts.

1 Introduction

Learning user preferences is an active research area in AI and it forms an important component of many modern applications in decision support systems, Web search, recommender systems and social media [16][17][19][20][24][29][48]. One important task is to present to the user a set of objects that match user context and are sorted in a decreasing order of preferences. More precisely, objects refer to anything of user interest (e.g. Web documents, shopping items, friends or banking services) whilst user contexts refer to anything that triggers the object response (e.g. submitted queries, user purchasing history or user profiles). Handcrafting a

*This work was done when authors are fully employed by Curtin University, WA, Australia.

good preference model is likely to be a complex and expensive task involving a great amount of domain knowledge. It is therefore desirable that preference models be automatically learnt from preference data that can be acquired from the users themselves for free, or with much less cost. Ideally, training data would be a collection of ranked sets of objects where each set contains objects related to a context. Unfortunately, providing a complete rank over a large set of objects is a mentally intensive task, since users need to simultaneously pay attention to all objects. As a result, modern large-scale preference data frequently consists of object ratings, where each object is rated by a degree of relevance with respect to the context.

There are two important implications with this practice. The first is that we must learn a ranking model indirectly from ratings, typically by inferring preference order by comparing the ratings of two objects[1]. Second, since ratings are usually drawn from a small set of integers, many objects may share the same rating, or equivalently they must share a tied rank. Thus, this poses a question of how to model set-level *preferences with ties*. The answer to this question is the focus of this paper.

Previous work on probabilistic tied preferences is fairly limited and typically considers pairwise comparisons [13][22][43][54]. There is no focus on the set-level preferences. We take an alternative approach by modelling objects with the same tie *with respect to the same context* as a partition, translating the problem into ranking or ordering these partitions. This problem transformation results in a combinatorial problem involving simultaneous set partitioning and subset ordering. For a given number of partitions, the order amongst them is a permutation of the partitions being considered, wherein each partition has objects of the same rank. Although this setting is general and flexible, we need to explore a super-exponential combinatorial state-space with unknown number of partitions and unknown ordering among them. To be more precise, the size of the state-space of N objects grows exponentially as $N!/(2\,(\ln 2)^{N+1})$ [40, pp. 396–397]. Learning and inference in such a a huge space are prohibitive, and this calls for efficient solutions which can be implemented in practice.

To this end, we introduce a probabilistic model that captures the *generative process* where a context-specific set is partitioned into subsets in a stagewise manner. More specifically, the user first chooses the first partition with elements of rank 1, then chooses the next partition from the remaining objects with elements ranked 2 and so on. The number of partitions then does not have to be specified in advance, and can be treated as a random variable. The joint distribution for each ordered partitioning can then be composed using a variant of the Plackett-Luce model [37][42] but at the level of partitions rather than objects. This approach offers two important benefits: (i) theoretically, it is interpretable in modelling user choices [37] in that users choose an object with probability proportional to its *utility*; and (ii) practically, this greatly reduces the size of state-space we have to deal with at each stage. While the stagewise state-spaces are still large, we can practically explore them using standard Markov Chain Monte Carlo (MCMC) techniques. In fact, we can train powerful models over a large-scale setting of hundreds of thousands objects using an ordinary PC. Another surprising result is that, with an appropriate choice of set utility functions (SUFs), we can learn the models in *linear* time. This is a great saving, is more efficient than existing pairwise models where

[1]This rating-to-rank conversion is not reversible since we cannot generally infer ratings from a ranking. First, the top rating for each context is always converted into rank 1 even if this is not the maximum score in the rating scale. Second, there are no gaps in ranking, while it is possible that we may rate the best object by 5 stars, but the second best by 3 stars.

the time complexity is generally quadratic in set sizes. We specify this construction as the *Probabilistic Model over Ordered Partitions* (PMOP).

We apply the proposed model for two practical problems. The first application is to rank Web document objects, where for each query we need to output a ranked list of documents in the decreasing order of relevance. In particular, we use a large-scale data supplied by Yahoo! [8] which consists of approximately $20K$ queries, and totally $470K$ documents. It is generally believed that each query captures user's intention and information needs, and as a result, by learning a ranking model, we learn collective preferences among users. We show that our results, both in terms of predictive performance and training time, are competitive against other well-known methods such as RankNet [6], (linear) Ranking SVM [31] and ListMLE [52].

Another application is movie recommendation, where each user has expressed their preferences in terms of ratings on a small number of movies, and the task is to recommend new movies they will like. In particular, we report our results on the MovieLens $10M$ data, which has lightly more than 10 millions ratings submitted by $71K$ users over a set of $10K$ movies. Here each user ID and their rating history represent a context, and the goal is to discover the hidden traits of each user to effectively rank new movies for them. For comparison, we evaluate our algorithms against the CoFi^{RANK} algorithm [51].

Our contribution, to the best of our knowledge, is the first to address the problem of modelling context-specific preferences with ties in its most generic form. We contribute by constructing a probabilistic model over ordered preference partitions and associated efficient inference and learning techniques. Furthermore, we show how to overcome the challenge of model complexity through the choice of suitable set utility functions, yielding a learning algorithm with linear time complexity, thus making the algorithm deployable in realistic settings. The novelty lies in the rigorous examination of probabilistic models over ordered partitions, extending earlier work in discrete choice theory [18][37][42]. The significance of the model is its potential for use in many applications. We demonstrate here two applications in learning-to-rank for Web retrieval and collaborative filtering for movie recommendation. Further, the model opens new potential applications, for example, novel types of clustering in which the clusters are automatically ordered.

The paper is organised as follows. Section 2 provides background for preference modelling, learning-to-ranking and collaborative filtering. The main theoretical contribution of this paper is presented in Section 3, whilst the practical contribution, with more detailed implementation, is described in Section 4. Section 5 reports experimental results for the two applications. Further theoretical implication is discussed in Section 6, followed by the conclusion section. Some detailed computation is reported in the appendix.

2 Background

In this section, we present necessary background related to our problem of learning context-specific preferences with ties and to two applications studied in this paper, namely Web document ranking and collaborative filtering.

Probabilistic Preference Modelling. The notion of preference is widely studied in many areas, including economics [37], statistics [39], artificial intelligence [2][12][17] [19][24][29]. A typical setting is pairwise preference, where an object x_i is preferred to another object x_j, denoted by $x_i \succ x_j$. Probabilistic approach often involves estimating the probability

that a particular preference order occurs, i.e. $P(x_i \succ x_j)$. One popular method is the Bradley-Terry model [3]: $P(x_i \succ x_j) = \phi(x_i)/\left(\phi(x_i) + \phi(x_j)\right)$ where $\phi(.) \in \mathbb{R}^+$ is a function indicating the utility (or the worth) of an object. This expression asserts that the probability of choosing an object over another is proportional to its utility. Later, this line of reasoning was generalised into the theory of discrete choice by Luce [37]. Under this assumption, we can extend the model to more than two objects, for example $P(x_i \succ \{x_j, x_k\}) = \phi(x_i)/\left(\phi(x_i) + \phi(x_j) + \phi(x_k)\right)$ is the probability that the object x_i is preferred over both x_j and x_k. In fact, the model can be generalised to obtain probability of a complete preference ordering $x_1 \succ x_2 \succ ... \succ x_N$ as follows

$$P(x_1 \succ x_2 \succ ... \succ x_N) = \prod_{i=1}^{N} \frac{\phi(x_i)}{\sum_{j=i}^{N} \phi(x_j)} \tag{1}$$

This model was first studied by Luce in [37] and subsequently by Plackett in [42], hence we shall refer to it as the Plackett-Luce model. The idea of this model is that, we select objects in a stagewise manner: Choose the first object among N objects with probability of $\phi(x_1)/\sum_{j=1}^{N} \phi(x_j)$, then choose the second object among the remaining $(N-1)$ objects with probability of $\phi(x_2)/\sum_{j=2}^{N} \phi(x_j)$ and so on until all objects are chosen. It can be verified that the distribution is proper, that is $P(x_1 \succ x_2 \succ ... \succ x_N) > 0$ and the probabilities of all possible orderings will sum to one.

Another approach to modelling complete ordering relies on the concept of rank distance between two rankings. The assumption is that there exists a *modal* ranking over all objects, and what we observe are ranks randomly distributed around the mode. The most well-known model is perhaps the Mallows [10][24][36][38], where the probability of a rank decreases exponentially with the distance from the mode

$$P(\boldsymbol{\pi}) \propto e^{-\lambda \tau(\boldsymbol{\pi}, \bar{\boldsymbol{\pi}})}$$

where $\boldsymbol{\pi}$ denotes an ordering, e.g. $\boldsymbol{\pi} = (x_1 \succ x_2 \succ ... \succ x_N)$, $\bar{\boldsymbol{\pi}}$ is the central ordering (the mode of the distribution), $\tau(.,.)$ is a rank-based distance function, and $\lambda > 0$ is model parameter. Popular distance measures include Kendall's *tau* and Spearman's *rho*. However, there are several drawbacks to this approach. First, assuming a single mode over all possible rankings is rather restrictive and therefore cannot handle situations where users form clusters resulting in multiple modes. Second, finding the mode is a challenging problem itself, since it would mean searching through a combinatorial space of $N!$ possibilities. Finally, we note in passing that there is a third approach that treats an ordering as an element of a symmetric group in the group theory [14][27]. However, since inference is complex, there has not been any sizable applications in practice following this approach.

Context-Specific Preferences and Tie Modelling. We are interested in the setting that each context is associated with a subset of objects which are more or less related to it. For example, the context can be a set of keywords that trigger responses from a typical search engine. Less explicitly, it can be a user with a rating history and will implicitly ask for a recommended item list from a shopping site. In machine learning under the setting of *label ranking* (e.g. see [29]), on the other hand, content would play the role of a context and labels the objects to be ranked. Usually the context-specific set is smaller than the set of all possible objects, thus leading to the problem of *incomplete* preferences. To deal

with this issue, naturally one would think of imputing missing objects, or integrating out the hidden preferences (e.g. in an Expectation-Maximisation setting). However, this is only practical when the set of all possible objects is finite and the data is dense. In Web search and collaborative filtering, however, these conditions do not hold since theoretically we have an infinite number of Web documents, and the rating data is often extremely sparse. (In our movie experiment setting, for example, each user picks only 1.1% number of all movies). This rules out the approach based on rank distances. Plackett-Luce based models, on the other hand, are highly applicable since we can simply ignore the missing objects for each context. As a result, we may have multiple preference models, one per context, sharing the same parameters[2].

Preference with ties has also been studied for several decades, mostly in statistical communities [13][22][43]. The general idea is to allocate some probability mass for the event of ties, e.g. $P(x_i \sim x_j) \neq 0$, where $\sim$ denotes indifference. For example, in the Davidson's method [13] the probability masses are defined as

$$P(x_i \succ x_j) = \frac{1}{Z_{ij}}\phi(x_i); \qquad P(x_i \sim x_j) = \frac{1}{Z_{ij}}\nu\sqrt{\phi(x_i)\phi(x_j)}$$

where $Z_{ij} = \phi(x_i) + \phi(x_j) + \nu\sqrt{\phi(x_i)\phi(x_j)}$ and $\nu \geq 0$ is model parameter controlling the contribution of ties. This model is quite intuitive in the sense that when tie occurs, both objects contribute equally to the probability. When $\nu = 0$, this reduces to the standard Bradley-Terry model. In a similar manner, Rao and Kupper [43] proposed the following model

$$P(x_i \succ x_j) = \frac{\phi(x_i)}{\phi(x_i) + \theta\phi(x_j)}; \quad P(x_i \sim x_j) = \frac{(\theta^2 - 1)\phi(x_i)\phi(x_j)}{[\phi(x_i) + \theta\phi(x_j)]\,[\theta\phi(x_i) + \phi(x_j)]}$$

where $\theta \geq 1$ is the parameter to control the contribution of ties. When $\theta = 1$, this also reduces to the standard Bradley-Terry model.

For ties among multiple objects, we can create a group of objects and work directly on groups. For example, let X_i and X_j be two sport teams, the pairwise team ordering can be defined using the Bradley-Terry model as $P(X_i \succ X_j) \propto \sum_{x \in X_i} \phi(x)$. An extension of the Plackett-Luce model to multiple groups has been discussed in [28]. However, this setting is inflexible since the the partitioning into groups must be known in advance, and the groups behave just like standard super-objects. A related work in [23] studies optimal partitioning, but it is limited to learning a single optimal partition out of a set.

To handling ties at the set level in the general form, we have proposed a probabilistic framework over ordered partitions (PMOP) in [47]. The framework assumes no prior knowledge of partitioning and ordering. Instead it models a generative stagewise process where each partition is selected at each stage. As a result, the PMOP is a Plackett-Luce model at the partition level, where partitions are automatically selected from the set of context-specific objects. The PMOP is learnt by maximising the data likelihood. However, this may not be optimal with respect to performance metrics for particular application domains. Second, for domains like Web document ranking, the feature interaction may have a strong effect on the behaviour of the model. In this paper, we further the study of PMOP by introducing

[2]This is also referred to as *parameter tying*.

a weighting scheme to bias the data likelihood towards performance metrics, and by investigating the second-order features interactions. For the application in movie recommendation, we evaluate PMOP on a new data set which is 2-order of magnitude larger than the previous work, and in a more interesting setting where only movies with minimal agreement among users are kept.

Web Document Ranking. Document ranking is the heart of any search engine. Originally, ranking was typically based on keyword matching, where a similarity measure between query keywords and the document is computed. Although there have been some fairly established measures including the *tf.idf* and *Okapi BM25*, these are not necessarily good indicators of how users judge the document relevancy and quality. Later development exploited the hyperlink structure of the Web, from which global ranking models such as PageRank [5] are used to compute the general quality of a Web page. In particular, PageRank demonstrates that modelling behaviour of Web surfers is critical. A more recent trend is to learn search preference models directly from labelled data. Typically, a set of preference indicators can be extracted from a given context, which may involve the user-submitted query, document content, link structure and search history and user profiles. The goal is to estimate a preference function that takes these indicators as input and outputs a real value score, from which documents are ranked. This trend is called *learning-to-rank* (LTR) [35] - a subfield of machine learning dealing with preference modelling. LTR algorithms can be characterised by their modelling nature - some are probabilistic: the RankNet [6], ListNet [7], BoltzRank [50] and ListMLE [52] and some are non-probabilistic: Ranking SVM [11][31] and RankBoost[19]. Within probabilistic models, the RankNet is essentially the Bradley-Terry model, and [7][52] are applications of the Plackett-Luce model. Tie handling within the pairwise setting has also been studied in [54].

Collaborative Filtering. Started in the early 1990s, collaborative filtering [44] is a highly successful approach to recommendation [1]. The applications range widely from movie recommendation, shopping item suggestion to financial advice and drug discovery. Recently, it has appeared from the Netflix challenge[3] that the two most effective methods are the neighbourhood-based (e.g. see [44][45]) and the latent trait discovery (e.g. see [32]). In the neighbourhood-based methods we estimate the similarity between any two users, or any two items. The premise is that two users who share interest in the past will continue the sharing in the future. On the other hand, in the latent trait methods we want to discover the hidden tastes of a user and profiles of an item. In particular, in the matrix factorisation approach to latent traits, a rating is approximated by

$$r_{ui} \approx \sum_{d=1}^{D} W_{ud} H_{di} \tag{2}$$

where W_{ud} and H_{di} are the d-th hidden trait of user u and item i, respectively. In this paper we follow the latent trait method, but the goal is to estimate the object user-specific utility rather than rating.

One important issue with the current practice in collaborative filtering is that it is largely based on rate prediction whilst the real goal is to rank items according to users' preferences. Although the rating could be used for ranking, it is better to directly address the rank problem

[3] http://www.netflixprize.com/

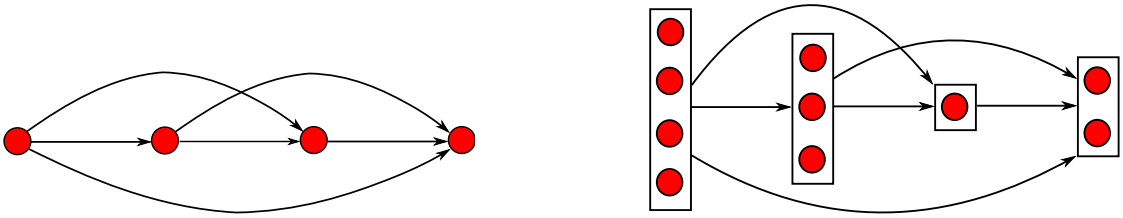

Figure 1: Complete ordering (left) versus subset ordering (right). For the subset ordering, the bounding boxes represents the subsets of elements of the same rank. Subset sizes are $4, 3, 1, 2$, respectively.

in the first place [34][46][51]. Our work is in preference modelling and thus deals with the ranking directly.

3 Modelling Sets with Ordered Partitions

In this section, we present our main contribution for solving the problem of modelling, learning and inference of context-specific preferences with ties. We first describe the problem in the most generic mathematical form, and then introduce a probabilistic framework to approach the problem. Efficient learning and inference techniques are presented to tackle the hyper-exponential state-space.

3.1 Problem Description

Let $X = \{x_1, x_2, \ldots, x_N\}$ be a collection of N objects related to a context. In a complete preference ordering setting, each object x_i is further assigned with an ordering index π_i , resulting in the ordered list of $\{x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_N}\}$ where $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_N)$ is a permutation over $\{1, 2, \ldots, N\}$. In our setting, π_1 is the rank of the first object, π_2 is the rank of second object and so on. Ideally, the training data should contain the complete ordering information $\boldsymbol{\pi}$ for all N objects; however, this task is not always possible for any non-trivial size N due to the labor cost involved[4]. Instead, in many situations, an object is *rated*[5] to indicate its perceived utility with respect to the context. The premise is that if two objects x_i and x_j are rated by numerical values r_i and r_j, respectively and $r_i \geq r_j$ then we have the preference order $x_i \succeq x_j$. This creates a scenario where more than one object will be assigned to the same rating – a situation known as '*ties*' in preference modelling. When we enumerate over each object x_i and put those with the same rating together, the set of N objects X can now be viewed as being divided into K partitions, each of which is assigned with a number to indicate its unique rank $k \in \{1, 2, .., K\}$. The ranks are obtained by sorting ratings associated with each partition in decreasing order.

Consider a more generic setting in which we know that each object will be rated against one of K *ordinal* values but do not know individual ratings. Assuming that all K values

[4]We are aware that clickthrough data can help to obtain a complete ordering, but the data may be noisy.

[5]We caution the confusion between 'rating' and 'ordering' here. Ordering is the process of sorting a set of objects in an increasing or decreasing order, whereas in 'rating' each object is given with a value indicating its preference.

have been used[6], this means that we have to consider all possible ways to split the set X into exactly K partitions, and then each time, *rank* those partitions from 1 to K wherein the k-th partition contains all objects rated with the same value k. Formally, for a given K and the order among the partitions $\boldsymbol{\sigma}$, we write the set $X = \{x_1, \ldots, x_N\}$ as a union of K partitions

$$X = \bigcup_{k=k}^{K} X_{\sigma_k} \tag{3}$$

where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_K)$ is a permutation over $\{1, 2, .., K\}$ and each partition X_k is a non-empty subset[7] of objects with the same rating r_k. These partitions are pairwise disjointed and having cardinality[8] range from 1 to N. For any $k_1 < k_2$, we write $X_{k_1} \succ X_{k_2}$ to denote that every object in X_{k_1} is preferred to all objects in X_{k_2}. More formally, we write

$$x_i \sim x_j \quad \forall x_i, x_j \in X_k; \qquad x_i \succ x_j \quad \forall x_i \in X_{k_1},\, x_j \in X_{k_2},\, k_1 < k_2. \tag{4}$$

See Figure 1 for a graphical illustration.

It is easy to see that when $K = N$, each X_k is a singleton, $\boldsymbol{\sigma}$ is now a complete permutation over $\{1, \ldots, N\}$ and the problem reduces exactly to the complete preference ordering mentioned earlier. To get an idea of the state space, it is not hard to see that there are $\left| \begin{matrix} N \\ K \end{matrix} \right| K!$ ways to partition and order X where $\left| \begin{matrix} N \\ K \end{matrix} \right|$ is the number of possible ways to divide a set of N objects into K partitions, otherwise known as *Stirling numbers of second kind* [49, p. 105]. If we consider all the possible values of K, the size of our state space is

$$\sum_{k=1}^{N} \left| \begin{matrix} N \\ k \end{matrix} \right| k! = \text{Fubini}\,(N) = \sum_{j=1}^{\infty} \frac{j^N}{2^{j+1}} \tag{5}$$

which is also known in combinatorics as the Fubini's number [40, pp. 396–397]. This is a super-exponential growth number. For instance, $\text{Fubini}\,(1) = 1$, $\text{Fubini}\,(3) = 13$, $\text{Fubini}\,(5) = 541$ and $\text{Fubini}\,(10) = 102,247,563$. Its asymptotic behaviour can also be shown [40, pp. 396–397] to approach $N!/(2\,(\ln 2)^{N+1})$ as $N \to \infty$. Note that $\ln(2) < 1$, and thus the Fubini number grows much faster than $N!$ - the complexity of complete ordering. Clearly, for unknown K, this presents a very challenging problem to inference and learning. In this paper, we shall present a stagewise construction to reduce the state-space at each stage significantly, and derive a generic MCMC-based approach to tackle this state-space explosion, and a specific parameterisation which leads to *linear* time in supervised learning settings.

3.2 Probabilistic Model over Ordered Partitions

Return to our problem, our task is now to model a distribution over the ordered partitioning of set X into K partitions and the ordering $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_K)$ among K partitions given in

[6] This can be done by removing unassigned values and decreasing K accordingly.

[7] Strictly speaking, a partition can be an empty set but we deliberately left out this case, because empty sets do not contribute to the probability mass of the model, and it does not match the real-world intuition of object's utility.

[8] More precisely, when the number of partitions K is given, the cardinality ranges from 1 to $N - K + 1$ since partitions are non-empty

Eq. (3):

$$P(X) = P(X_{\sigma_1}, \ldots, X_{\sigma_K}) \tag{6}$$

A two-step approach has been given thus far: first X is partitioned in any arbitrary way so long as it creates K partitions and then these partitions are ordered, result in a ranking index vector $\boldsymbol{\sigma}$. This description is generic and one can proceed in different ways to further characterise Eq. (6). We present here a generative, multistage view to this problem so that it lends naturally to the specification of the distribution in Eq. (7): First, we construct a subset X_1 from X by collecting all objects which have the largest ratings. If there are more elements in the the remainder set $\{X \setminus X_1\}$ to be selected (e.g. $\{X \setminus X_1\} \neq \emptyset$), we construct a subset X_2 from $\{X \setminus X_1\}$ whose elements have the second largest ratings. This process continues until there is no more object to be selected.[9] An advantage of this view is that the resulting total number of partitions K_σ is automatically generated, no need to be specified in advance and can be treated as a random variable. If our data truly contains K partitions then K_σ should be equal to K. Using the chain rule, we write the joint distribution over K_σ ordered partitions as

$$\begin{aligned} P(X_1, \ldots, X_{K_\sigma}) &= P(X_1) \prod_{k=2}^{K_\sigma} P(X_k \mid X_1, \ldots, X_{k-1}) \\ &= P(X_1) \prod_{k=2}^{K_\sigma} P(X_k \mid X_{1:k-1}) \end{aligned} \tag{7}$$

where we have used $X_{1:k-1} = \{X_1, \ldots, X_{k-1}\}$ for brevity.

It remains to specify the local distribution $P(X_k \mid X_{1:k-1})$. Let us first consider what choices we have after the first $(k-1)$ partitions have been selected. It is clear that we can select any objects from the remainder set $\{X \setminus X_{1:k-1}\}$ for our next partition k-th. If we denote this remainder set by $R_k = \{X \setminus X_{1:k-1}\}$ and $N_k = |R_k|$ is the number of remaining objects, then our next partition X_k is a subset of R_k; furthermore, there is precisely $\left(2^{N_k} - 1\right)$ such non-empty subsets. Using the notation 2^{R_k} to denote the *power set* of the set R_k, i.e, 2^{R_k} contains all possible non-empty subsets[10] of R, we are ready to specify each local conditional distribution in Eq. (7) as:

$$P(X_1) = \frac{1}{Z_1} \Phi_1(X_1); \qquad P(X_k \mid X_{1:k-1}) = \frac{1}{Z_k} \Phi_k(X_k) \tag{8}$$

where $\Phi_k(S) \in \mathbb{R}^+$ is a permutation-invariant[11] *set utility function* (SUF) defined over a subset or partition S, and $Z_k = \sum_{S \in 2^{R_k}} \Phi_k(S)$ is the normalising constant. We term our model *Probabilistic Model over Ordered Partition* (*PMOP*). The following proposition ensures that the distribution constructed in this way is proper.

[9] This process resembles the generative process of Plackett-Luce discrete choice model [37][42], except we apply on partitions rather than single element. It clear from here that Plackett-Luce model is a special case of ours wherein each partition X_k reduces to a singleton.

[10] The usual understanding would also contain the empty set, but we exclude it in this paper.

[11] i.e., the function value does not depend on the order of elements within the partition.

Proposition 1 *Denote by $\Im_X$ a particular partitioning and ordering scheme of the set X, and by $P(\Im_X)$ the corresponding probability distribution in Eq (6). Then the distribution constructed in Eq (8) is proper, i.e.*

1. $P(\Im_X) > 0$ *for all* $\Im_X$, *and*
2. $\sum_{\Im_X} P(\Im_X) = 1$.

Sketch of proof: The first condition is trivially satisfied since SUFs are positive. Now assume that X is split into the first partition X_1 and the rest $X_{\neg 1}$, that is, $X = \{X_1, X_{\neg 1}\}$ where $X_{\neg 1}$ may consist of one or more partitions with $X_1 \prec X_{\neg 1}$. Since X_1 is already a partition we have $\sum_{X_1} P(X_1) = 1$ due to Eq (8). The key is to note that if $P(\Im_{X_{\neg 1}} \mid X_1)$ is proper for all X_1, then $P(\Im_X)$ is also proper:

$$\sum_{\Im_X} P(\Im_X) = \sum_{X_1} P(X_1) \sum_{\Im_{X_{\neg 1}}} P(\Im_{X_{\neg 1}} \mid X_1) = 1.$$

We note that if $X_{\neg 1}$ is a partition, then $\sum_{\Im_{X_{\neg 1}}} P(\Im_{X_{\neg 1}} \mid X_1) = P(X_{\neg 1} \mid X_1) = 1$ by construction. Otherwise, we then work backward and assume the decomposition $X_{\neg 1} = \{X_2, X_{\neg 1:2}\}$ for any partition X_2 with $X_2 \succ X_{\neg 1:2}$. Again, if $\sum_{\Im_{X_{\neg 1:2}}} P(\Im_{X_{\neg 1:2}} \mid X_{1:2})$ is proper for all $\{X_1, X_2\}$, then $P(\Im_{X_{\neg 1}} \mid X_1)$ is proper. After at most $(N-1)$ splitting steps, we are left with a single object, which is obviously a partition itself ♣

To fully specify the PMOP, we need to define a parametric form of the SUFs. We defer the details until the next application section and assume for now that some form has been defined. Learning the model parameters can be carried out by maximising the data likelihood. Using Eqs. (8) and (7), the log-likelihood function and its gradient, without explicit mention of the model parameters, are:

$$\mathcal{L} = \log P(X_1) + \sum_{k=2}^{K_\sigma} \log P(X_k \mid X_{1:k-1}) \tag{9}$$

$$\partial\mathcal{L} = \sum_{k=1}^{K_\sigma} \partial \log \Phi_k(X_k) - \sum_{k=1}^{K_\sigma} \left\{ \sum_{S \in 2^{R_k}} P(S \mid X_{1:k-1}) \, \partial \log \Phi_k(S) \right\} \tag{10}$$

Clearly for learning, we need to compute both the $\mathcal{L}$ (for monitoring) and $\partial\mathcal{L}$ (for gradient-based optimisation). Computing the data log-likelihood reduces to evaluating $P(X_k \mid X_{1:k-1})$, which depends on Z_k as in Eq. (8). Likewise, the gradient of the log-likelihood depends on model expectation $\sum_{S \in 2^{R_k}} P(S \mid X_{1:k-1}) \, \partial \log \Phi_k(S)$. Unfortunately, these two quantities cannot generally be evaluated exactly except for trivial cases where the set X contains only a few objects.

In the rest of this section, we will concentrate our effort on solving this problem. In particular, in Section 3.3, we present MCMC-based sampling methods for approximate evaluation in manageable time. In Section 3.4, we describe a specific parameterisation which leads to exact evaluation in linear time.

3.3 MCMC-based Inference and Learning for General PMOP

In this subsection, we first discuss how random samples can be used in (i) evaluating the normalisation constant (e.g. used in computing the data log-likelihood), (ii) estimating model expectations (e.g. used in computing log-likelihood gradient), and (iii) learning with stochastic gradient. Then we describe the details of two MCMC-based sampling procedures: Gibbs sampling and Metropolis-Hastings sampling.

3.3.1 Inference and Learning Tasks

Let us first assume for now that there exists a sampling procedure that draws *set* samples from $P\left(S \mid X_{1:k-1}\right)$, the inference and learning tasks are then:

Evaluating normalising constant. To estimate the normalisation constant Z_k, we employ an efficient procedure called Annealed Importance Sampling (AIS) proposed recently [41]. More specifically, AIS introduces the notion of inverse-temperature $\tau \in [0, 1]$ into the model, that is

$$P_\tau\left(S \mid X_{1:k-1}\right) = \frac{1}{Z_k(\tau)} \Phi_k\left(S\right)^\tau$$

where $Z_k = \sum\limits_{S \in 2^{R_k}} \Phi_k(S)^\tau$. Let $\{\tau_t\}_{t=0}^T$ be the sequence of (slowly) increasing temperature $0 = \tau_0 < \tau_1 ... < \tau_T = 1$. At $\tau_0 = 0$, the model distribution is uniform and $Z_k(0) = 2^{|N_k|} - 1$ which is simply the number of all possible non-empty subsets from $|N_k|$ remaining objects. At $\tau_T = 1$, the desired distribution is obtained.

At each step t, a sample $S^{(t)}$ is drawn from the distribution $P_{\tau_{t-1}}\left(S \mid X_{1:k-1}\right)$. The final weight after the (inverse) annealing process is computed as

$$\omega = \prod_{t=1}^{T} \Phi_k\left(S^{(t)}\right)^{\tau_t - \tau_{t-1}}$$

The above procedure is repeated R times. Finally, the normalisation constant at $\tau = 1$ is estimated to be $Z_k(1) \approx Z_k(0) \left(\sum_{r=1}^{R} \omega^{(r)} / R\right)$.

Computing model expectation. The model expectation can be approximated by

$$\sum_{S \in 2^{R_k}} P\left(S \mid X_{1:k-1}\right) \partial \log \Phi_k\left(S\right) \approx \frac{1}{n} \sum_{l=1}^{n} \partial \log \Phi_k\left(S^{(l)}\right)$$

where $\{S^{(l)}\}_{l=1}^n$ are set samples returned by the sampling procedure.

Stochastic gradient learning. Given an approximate model expectation, the gradient is not exact but rather stochastic and thus standard optimisation routines cannot be used. Fortunately, it has been shown that stochastic gradient ascent can still maximise the data likelihood if the learning rate is sufficiently small [53][26]. There are two general ways to combine the sampling procedure and parameter update. One is to keep multiple Markov chains in parallel and alternate between parameter update and sampling [53]. The parameter

update is likely to change the model slightly and the sampler can still run as if the model is still the same. Another method is to restart the chains from the data samples and run for a few steps before every updating [26]. In this paper, we follow the latter since it is (i) easier to implement and we do not have to maintain previous samples, (ii) the Markov chains are restarted at every updating step and thus may avoid being trapped in low density regions.

3.3.2 Gibbs Sampling

We first note that the problem of sampling sets from $P\left(S \mid X_{1:k-1}\right)$ can be viewed as exploring a *fully-connected Markov random field with binary variables* [21]. At stage k, each remaining object in $\{X \setminus X_{1:k-1}\}$ is attached with a binary variable whose states[12] are either `selected` or `notselected`. There will be $2^{N_k} - 1$ joint states in the random field, where we recall that N_k is the total number of remaining objects after the $(k-1)$-th stage. The SUF $\Phi_k(X_k)$ is now the utility function of all selected variables. We then cyclically pick an object from $\{X \setminus X_{1:k-1}\}$ and then randomly select it with probability of

$$P_k(\texttt{selected} \mid x) = \frac{\Phi_k(X_k^{+x})}{\Phi_k(X_k^{+x}) + \Phi_k(X_k^{-x})}$$

where $\Phi_k(X_k^{+x})$ is the utility of the currently selected subset X_k if x is included and $\Phi_k(X_k^{-x})$ if not. Since samples collected in this way are highly correlated, we only need to retain a sample after every fixed number of passes through all remaining objects. Other samples are discarded. This procedure is known as *thinning*, and it helps to reduce the computational overhead significantly.

The main advantage of this method lies in its simplicity with no tuning parameters at the tradeoff of the speed to achieve independent samples. The pseudo code for the Gibbs routine performed at k-th stage is summarised in Algorithm 1.

3.3.3 Metropolis-Hastings Sampling

The idea is to consider a subset $S \in 2^{R_k}$ itself as a sampling state [25] and randomly move from one sample state to another. The move from S to S' is accepted with the following probability

$$\min\left\{1, \frac{P\left(S' \mid X_{1:k-1}\right)}{P\left(S \mid X_{1:k-1}\right)} \frac{Q(S|S')}{Q(S'|S)}\right\} = \min\left\{1, \frac{\Phi_k(S')}{\Phi_k(S)} \frac{Q(S|S')}{Q(S'|S)}\right\}$$

where $Q(S'|S)$ is the proposal transition probability which depends on specific proposal design, and we have canceled the normalising constant Z_k using Eq. (8). For simplicity, assume that all the sets are uniformly randomly selected from $\{X \setminus X_{1:k-1}\}$, then this acceptance rate is simplified to

$$\min\left\{1, \frac{\Phi_k(S')}{\Phi_k(S)}\right\}$$

Again, to reduce correlation, we only need to retain a sample after a number of steps and discard others. The pseudo code for the Metropolis-Hastings routine is summarised in Algorithm 2.

[12] Note that these states are defined for the Markov random field under current consideration only.

Algorithm 1 Gibbs sampling for PMOP in general case.

Input: input parameters, sample collection schedule.

1. Randomly choose an initial subset X_k.
2. **Repeat until** stopping criteria met:
   (a) For each remaining object x at stage k, randomly select the object with the probability
   $$\frac{\Phi_k(X_k^{+x})}{\Phi_k(X_k^{+x}) + \Phi_k(X_k^{-x})}$$
   where $\Phi_k(X_k^{+x})$ is the utility of the currently selected subset X_k if x is included and $\Phi_k(X_k^{-x})$ is when x is not.
   (b) Collect samples according to schedule.

Output: output set samples.

Algorithm 2 Metropolis-Hastings sampling for PMOP in general case.

Input: input parameters, sample collection schedule.

1. Randomly choose an initial subset X_k.
2. **Repeat until** stopping criteria met:
   (a) Randomly choose number of objects m, subject to $1 \leq m \leq N_k$.
   (b) Randomly choose m distinct objects from the remaining set $R_k = X \backslash X_{1:1:k-1}$ to construct a new partition denoted by S.
   (c) Set $X_k \leftarrow S$ with the probability of
   $$\min\left\{1, \frac{\Phi_k(S)}{\Phi_k(X_k)}\right\}$$
   (d) Collect samples according to schedule.

Output: output set samples.

3.4 Set Utility Functions and Their Decomposition

Up until now we have not made any assumption of the SUF $\Phi_k(X_k)$ although it plays a crucial role in our PMOP. In general, objects within a partition X_k interact with each other, and correctly specifying the SUF would depend critically on application requirements. In this subsection, we limit to a special setting where SUF is an aggregation function of individual utilities. We first discuss some desirable practical properties of the function and then present a special parameterisation that leads to very efficient computation of data likelihood and its gradient. See further discussion on Section 6.

3.4.1 SUFs as Aggregation

Although it is necessary to correctly model the partitioning and ordering process at training time, it is in many cases more practical to obtain a complete ordering of objects at prediction time. This is because a complete order is more interpretable to users so that they may decide to select only a few top ranked objects. Another practical benefit is that, it helps to reduce the size of the state space significantly, from Fubini (N) (simultaneous partitioning and ordering) to $N!$ (ordering only). One of the most efficient ways is perhaps to assign a *utility* value to each object, and then order the objects accordingly using standard sort algorithms. This costs $\mathcal{O}(N \log N)$ time on average. The question now is how to estimate the object utility during the training phase. Denote by $\phi(x) \in \mathbb{R}^+$ the utility of object x, the SUF arising from Eq. (8) can then be rewritten as

$$\Phi_k(X_k) = h\{\phi(x) | x \in X_k\}$$

where $h\{\cdot\}$ denotes the aggregation from a set of individual functions.

Regularising the set size. We can impose our prior knowledge about the set sizes. For example, we know that in the Web search setting, there are far more irrelevant documents than highly relevant ones. Thus, we can further introduce a bias on the set size into $\Phi_k(\cdot)$. In the case of lacking specific knowledge, however, it may be more robust to regularise its dependency on the set size[13]. One way to achieve this is to enforce the constraint:

$$\min\{\phi(x) | x \in X_k\} \leq \Phi_k(X_k) \leq \max\{\phi(x) | x \in X_k\}$$

In fact, there are many common aggregation functions satisfying this condition, for example:

- Arithmetic mean $\Phi_k(X_k) = \frac{1}{|X_k|} \sum_{x \in X_k} \phi(x)$
- Geometric mean $\Phi_k(X_k) = \left\{ \prod_{x \in X_k} \phi(x) \right\}^{\frac{1}{|X_k|}}$
- Minimum $\Phi_k(X_k) = \min\{\phi(x) | x \in X_k\}$
- Maximum $\Phi_k(X_k) = \max\{\phi(x) | x \in X_k\}$

[13] Regularising the dependency on set size may also be more interpretable: if the set utility is the sum of individual utilities, one may simply choose all the objects to maximise the utility.

Monotonic property. Naturally, we would expect that if X_k contains worthy objects, then $\Phi_k(X_k)$ should return a large value, mathematically put:

$$\Phi_k(S \cup x_i) \geq \Phi_k(S \cup x_j) \text{ if } \phi(x_i) \geq \phi(x_j)$$

for any $\{S, x_i, x_j\}$, where S can be an empty set. It can be verified that the four types of aggregation function listed above satisfy this condition.

3.4.2 Arithmetic Mean SUF

Now we focus on a specific type of aggregation function - the arithmetic mean:

$$\Phi_k(X_k) = \frac{1}{|X_k|} \sum_{x \in X_k} \phi(x) \tag{11}$$

Given this form, the local normalisation factor represented in the denominator of Eq. (8) can now be efficiently represented as the sum of all weighted sums of objects. Since each object x in the remainder set R_k participates in the *same* additive manner towards the construction of the denominator in Eq. (8), it must admit the following form[14]:

$$\sum_{S \in 2^{R_k}} \Phi_k(S) = \sum_{S \in 2^{R_k}} \frac{1}{|S|} \sum_{x \in S} \phi(x) = C \sum_{x \in R_k} \phi(x) \tag{12}$$

where C is some constant and its exact value is not essential under a maximum likelihood parameter learning treatment (readers are referred to Appendix A.1 for the computation of C, which happens to be $(2^{N_k}-1)/N_k$). To see this, let us substitute Eqs. (11) and (12) into Eq. (8):

$$\begin{aligned} \log P(X_k \mid X_{1:k-1}) &= \log \frac{\Phi_k(X_k)}{\sum_{S \in 2^{R_k}} \Phi_k(S)} = \log \frac{1}{C|X_k|} \frac{\sum_{x \in X_k} \phi(x)}{\sum_{x \in R_k} \phi(x)} \\ &= \log \frac{\sum_{x \in X_k} \phi(x)}{\sum_{x \in R_k} \phi(x)} - \log C|X_k| \end{aligned} \tag{13}$$

Since $\log C|X_k|$ is a constant w.r.t the parameters used to parameterise the utility functions $\phi_k(\cdot)$, it does not affect the gradient of the log-likelihood. It is also clear that maximising the likelihood given in Eq. (7) is equivalent to maximising each local log-likelihood function given in Eq. (13) for each k. Discarding the constant term in Eq. (13), we re-write it in this simpler form:

$$\log P(X_k \mid X_{1:k-1}) = \log \sum_{x \in X_k} g_k(x \mid X_{1:k-1}) \tag{14}$$

$$\text{where } \; g_k(x \mid X_{1:k-1}) = \frac{\phi(x)}{\sum_{x \in R_k} \phi(x)}$$

[14] To illustrate this intuition, suppose the remainder set is $R_k = \{a, b\}$, hence its power set, excluding $\emptyset$, contains 3 subsets $\{a\}, \{b\}, \{a, b\}$. Under the arithmetic mean assumption, the denominator in Eq. (8) becomes $\phi(r_a) + \phi(r_b) + \frac{1}{2}\{\phi(r_a) + \phi(r_b)\} = (1 + \frac{1}{2}) \sum_{x \in \{a,b\}} \phi(r_x)$. The constant term is $C = \frac{3}{2}$ in this case.

Pairwise models	PMOP
$\max\{\mathcal{O}(N^2), \mathcal{O}(NF)\}$	$\mathcal{O}(NF)$

Table 1: Learning complexity of models, where F is the number of unique features. For pairwise models, see Appendix A.2 for the details.

Efficient gradient evaluation using dynamic programming. We show that the gradient-based learning complexity can be linear in N. To see how, let us introduce an auxiliary array $\left\{a_k = \sum_{x \in R_k} \phi(x)\right\}_{k=1}^{K_\sigma}$. We start from the last subset, where $a_{K_\sigma} = \sum_{x \in X_{K_\sigma}} \phi(x)$, and proceed backwards as $a_k = a_{k+1} + \sum_{x \in X_k} \phi(x)$ for $k < K_\sigma$. Clearly $\{a_1, a_2, .., a_{K_\sigma}\}$ can be computed in N time. Thus, $g_k(\cdot)$ in Eq. (14) can also be computed linearly via the relation $g_k(x) = \phi(x)/a_k$. This also implies that the total log-likelihood can also computed linearly in N.

Furthermore, the gradient of log-likelihood function can also be computed linearly in N. Given the likelihood function in Eq. (7), using Eq. (14), the log-likelihood function and its gradient, without explicit mention of the parameters, can be shown to be[15]

$$\mathcal{L} = \log P(X_1, \ldots, X_{K_\sigma}) \tag{15}$$

$$= \sum_{k=1}^{K_\sigma} \log \sum_{x \in X_k} g_k(x \mid X_{1:k-1}) = \sum_{k=1}^{K_\sigma} \log \sum_{x \in X_k} \frac{\phi(x)}{a_k}$$

$$\partial\mathcal{L} = \sum_k \partial \log \sum_{x \in X_k} \phi(x) - \sum_k \partial \log a_k \tag{16}$$

$$= \sum_k \frac{\sum_{x \in X_k} \partial\phi(x)}{\sum_{x \in X_k} \phi(x)} - \sum_k \frac{1}{a_k} \sum_{x \in R_k} \partial\phi(x) \tag{17}$$

It is clear that the first summation over k in the RHS of the last equation takes exactly N time since $\sum_{k=1}^{K} |X_k| = N$. For the second summation over k, it is more involved because both k and $|R_k|$ can possibly range from 1 to N, so direct computation will cost at most $N(N-1)/2$ time. Similar to the case of a_k, we now maintain a 2-D auxiliary array[16] $\left\{\mathbf{b}_k = \sum_{x \in R_k} \partial\phi(x)\right\}_{k=1}^{K_\sigma}$. Again, we start from the last subset, where $\mathbf{b}_{K_\sigma} = \sum_{x \in X_{K_\sigma}} \partial\phi(x)$, and proceed backward as $\mathbf{b}_k = \mathbf{b}_{k+1} + \sum_{x \in X_k} \partial\phi(x)$ for $k < K_\sigma$. Thus, $\{\mathbf{b}_1, \mathbf{b}_2, .., \mathbf{b}_{K_\sigma}\}$ and therefore the gradient $\partial\mathcal{L}$, can be computed in NF time, where F is the number of parameters.

Table 1 summarises the complexity of the PMOP in comparison with pairwise models (see also Appendix A.2 for details).

3.5 Enhancing Learning by Rank Weighting

One drawback of Plackett-Luce-style models is that they treat all objects in an equal manner. However, in many practical settings such as Web search, most objects are irrelevant, and incorporating them all would be sub-optimal. Here we propose a way to reduce the impact of lowly ranked objects through scaling. In particular, we modify Eq. (7) as

[15] To be more precise, for $k = 1$ we define $X_{1:0}$ to be $\emptyset$.

[16] This is 2-D because we also need to index the parameters as well as the subsets.

$$P^*\left(X_1,\ldots,X_{K_\sigma}\right) \;=\; P^{\alpha_1}\left(X_1\right)\prod_{k=2}^{K_\sigma} P^{\alpha_k}\left(X_k \mid X_{1:k-1}\right) \tag{18}$$

where $P^*\left(X_1,\ldots,X_{K_\sigma}\right)$ is the unnormalised probability and $\alpha_1 \geq \alpha_2 \geq ... \geq \alpha_{K_\sigma} \geq 0$ are rank weights. Thus, learning can be carried out by maximising the unnormalised probability

$$\begin{aligned}\mathcal{L}^* &= \log P^*\left(X_1,\ldots,X_{K_\sigma}\right)\\ &= \alpha_1 \log P\left(X_1\right) + \sum_{k=2}^{K_\sigma} \alpha_k \log P\left(X_k \mid X_{1:k-1}\right)\end{aligned}$$

4 Applications with PMOP

In this section, we describe the more detailed model specifications of PMOP in two applications, namely, Web document ranking and collaborative filtering.

4.1 Web Document Ranking

In this application, also known as learning-to-rank, the goal is to present to the user a list of related objects and their *ordering*[17] given a submitted query. The context can be constructed from the query, user browsing history or search behaviour. For the purpose of this application, we assume that for an unseen query q a list of $X^q = \left\{x_1^q,\ldots,x_{N_q}^q\right\}$ objects related to q is given[18], and for each query-object pair (q,x), a feature vector $x^q \in \mathbb{R}^F$ is readily available. The task is then to rank these objects in decreasing order of user preferences. Enumerating over all possible orderings take an order of $N_q!$ time. Instead we would like to establish a *value function* $f(x^q,w) \in \mathbb{R}$ for the query q and each object x returned, where w is now introduced as the parameter. Sorting can then be carried out much more efficiently in the complexity of $N_q \log N_q$ instead of $N_q!$. The function specification can be a simple linear combination of features or more complicated forms, such as multilayer neural networks.

In the practice of learning-to-rank, the dimensionality of feature vector x^q often remains the same across all queries, and since it is observed, we use PMOP described earlier to parameterise a query-specific model over the set of returned objects X^q as follows.

$$P\left(X^q|w\right) \;=\; P(X_1^q, X_2^q, ..., X_{K_\sigma}^q \mid w) = P(X_1^q \mid w)\prod_{k=2}^{K_\sigma} P(X_k^q \mid X_{1:k-1}^q, w) \tag{19}$$

We can see that Eq. (19) has exactly the same form of Eq. (7) specified for PMOP, but applied instead on the query-specific set of objects X^q and additional parameter w. During training, each query-object pair is labelled by a relevance score, which is typically an integer

[17] We note a confusion that may arise here is that, although during training each training query q is supplied with a list of related objects and their *ratings,* during the testing phase the system still needs to return an ordering over the list of related objects for an unseen query.

[18] In document querying, for example, the list may consist of all documents which contain one or more query words

from the set $\{0, .., M\}$ where 0 means the object is irrelevant w.r.t the query q, and M means the object is highly relevant[19]. The value of M is typically much smaller than N_q, thus, the events of *ties* occur frequently. In a nutshell, for each training query q and its rated associated list of objects, a PMOP is created. *The important parameterisation to note here is that the parameter w is shared across all queries*; and thus, enabling ranking for unseen queries in the future.

Using the value function $f(x, w)$ we specify the individual utility function $\phi(\cdot)$ in the exponential form: $\phi(x, w) = \exp\{f(x, w)\}$. For simplicity, assume further that the value function has the linear form $f(x^q, w) = w^\top x^q$. In what follows we choose two specific forms of $\Phi_k\left(X_k^q\right)$: one as a geometric mean and another as an arithmetic mean of individual utilities (Subsection 3.4). The geometric mean setting illustrates a particular case where we need to employ MCMC-techniques while the arithmetic mean setting leads to efficient computation.

4.1.1 Geometric Mean SUFs

Recall from Subsection 3.4 that the geometric mean SUF has the following form

$$\Phi_k\left(X_k^q\right) = \left(\prod_{x \in X_k^q} \phi(x)\right)^{\frac{1}{|X_k^q|}} = \exp\left\{\frac{1}{|X_k^q|} \sum_{x \in X_k^q} f(x, w)\right\} = \exp\left\{w^\top \bar{x}_k^q\right\} \tag{20}$$

where $\bar{x}_k^q = \frac{1}{|X_k^q|} \sum_{x \in X_k} x^q$ is simply the mean feature vector over the set X_k. The gradient of the log-likelihood function can be shown to be:

$$\frac{\partial \log P\left(X_k^q \mid X_{1:k-1}^q\right)}{\partial w} = \bar{x}_k^q - \sum_{S_k \in 2^{R_k^q}} P\left(S_k \mid X_{1:k-1}^q\right) \bar{s}_k^q$$

where $\bar{s}_k^q = \frac{1}{|S_k|} \sum_{x \in S_k} x^q$. The quantity $P\left(X_k^q \mid X_{1:k-1}^q\right)$ can be interpreted as the probability that the subset X_k^q is chosen out of all possible subsets at stage k, and $\bar{x}_k$ is the centre of the chosen subset.

The expectation $\sum_{S_k} P(S_k \mid X_{1:k-1}^q)\bar{s}_k$ is computationally expensive to evaluate, since there are $2^{N_k} - 1$ possible subsets. Thus, we resort to MCMC techniques (see Section 3.3) and the parameter is stochastically updated as follows

$$w \leftarrow w + \eta \frac{1}{|\mathcal{D}|} \sum_{q=1}^{|\mathcal{D}|} \sum_k \left(\bar{x}_k^q - \frac{1}{n} \sum_{l=1}^{n} \bar{s}_k^{(l)}\right)$$

where $|\mathcal{D}|$ is the number of training queries, $\bar{s}_k^{(l)}$is the centre of the subset sampled at iteration l, and $\eta > 0$ is the learning rate, and n is the number of samples. Typically we choose n to be small, e.g. $n = 1, 2, 3$.

[19]Note that generally $K \leq M + 1$ because there may be gaps in rating scales for a specific query.

4.1.2 Arithmetic Mean Set SUFs

Recall from Subsection 3.4 that the SUF is simply the mean of local utilities:

$$\Phi_k\left(X_k^q\right) = \frac{1}{|X_k^q|}\sum_{x\in X_k^q}\exp\left\{f(x,w)\right\} \tag{21}$$

The gradient of the log-likelihood function can also be computed efficiently:

$$\frac{\partial \log P\left(X_k^q \mid X_{1:k-1}^q\right)}{\partial w} = \sum_{x\in X_k^q}\frac{\phi_k(x,w)x}{\sum_{x\in X_k^q}\phi_k(x,w)} - \sum_{x\in R_k^q}\frac{\phi_k(x,w)x}{\sum_{x\in R_k^q}\phi_k(x,w)}$$

4.1.3 Second-Order Features

In the case of Web document ranking, features are often precomputed based on prior knowledge about: (i) *similarity* measures between a query and a related document, e.g. cosine between the word vectors with *tf.idf* weighting scheme, and (ii) *quality* measures of a document, e.g. sources, authority, language model, HITS or PageRank scores. One aspect still largely ignored is the interaction between those features, and how much they would contribute to the final estimation of the ranking function. On the other hand, some interaction can be totally unrelated to ranking task - this begs a question of feature selection. Since the number of these second-order features can be both large and dense - we may not afford to run a throughout feature selection procedure. Here we propose a two-step procedure:

1. *Feature generation*: performing a Cartesian product to generate all second-order features, and

2. *Feature selection*: first computing a correlation score between a generated feature and the relevance score given in training data. We then select only those features whose correlation score is larger than a threshold $\rho > 0$. In this paper, we use absolute Pearson's correlation measure

$$\left|\frac{\sum_d(y_{ad}-\bar{y}_a)(r_d-\bar{r})}{\sqrt{\sum_d(y_{ad}-\bar{y}_a)^2}\sqrt{\sum_d(r_d-\bar{r})^2}}\right|$$

   where a is the feature index and d is the document index, $\bar{y}_a$ is the mean of feature a and $\bar{r}$ is the mean of relevance score over training data. Note that $\sum_d$ means summing over all documents in the training data.

4.2 Collaborative Filtering

We now present an application of our PMOP in collaborative filtering. Recall that in collaborative filtering, we are given a set of users, each of whom has expressed preferences over a set of items. Let A be the number of users and B the number of items. To facilitate the interaction between a user u and an item i, the item utility function can be chosen as follows

$$\phi_k\left(x_i,u\right) = \exp\{f(i,u)\}, \text{ where } f(u,i) = \sum_{d=1}^{D} W_{ud}H_{di} \tag{22}$$

where $W \in \mathbb{R}^{A\times D}$, $H \in \mathbb{R}^{B\times M}$ and D is the hidden dimensionality (typically $D < \min\{A, B\}$). This bears some similarity with the matrix factorisation in Eq. (2), but we do not aim to approximate the rating per se.

Different from the case of document ranking, the feature vector for each item (H_{1i}, H_{2i}, .., H_{Di}) is not given and must be discovered from the training data. Recall that each user in the training data has expressed preferences over some seen items. By maximising data likelihood, we learn the parameter matrices W and H. However, the log-likelihood function is no longer concave in both W and H, although it is still concave in either W or H.

Denote by $L_k^u = \log P\left(X_k^u \mid X_{1:k-1}^u\right)$. For arithmetic mean SUFs, the gradient of the log-likelihood reads:

$$\frac{\partial L_k^u}{\partial W_{ud}} = \sum_{i\in X_k^u} \frac{\phi_k(i,u)H_{di}}{\sum_{j\in X_k^u}\phi_k(j,u)} - \sum_{i\in R_k^u} \frac{\phi_k(i,u)H_{di}}{\sum_{j\in R_k^u}\phi_k(j,u)}$$

$$\frac{\partial L_k^u}{\partial H_{di}} = \phi_k(i,u)W_{ud}\left(\frac{\delta[i\in X_k^u]}{\sum_{j\in X_k^u}\phi_k(j,u)} - \frac{\delta[i\in R_k^u]}{\sum_{j\in R_k^u}\phi_k(j,u)}\right)$$

where $\delta[\cdot]$ is the indicator function.

As the parameters are user-dependent (e.g. the number of rows in W grows with the number of users), and the log-likelihood is not convex, we need to carefully regularise the learning process. First, notice that since training data is often in the form of ratings, and converting from rating to ranking is lossy, it may be useful to bias the item utility toward ratings, and at the same time, to promote ranking with PMOP. Denote by r_{ui} the rating which user u has given to item i. The regularised log-likelihood over all users is

$$\hat{\mathcal{L}} = \sum_u \left\{\beta\mathcal{L}^u(W,H) + \frac{1}{2}(1-\beta)\sum_{i\in X^u}\left(r_{ui} - f\left(i,u\right)\right)^2\right\} + \lambda_1\left\|W\right\|_2^2 + \lambda_2\left\|H\right\|_2^2 \tag{23}$$

where $\mathcal{L}^u = \sum_k L_k^u$, $\beta \in [0,1]$ is a parameter to control the bias strength, $\lambda_1, \lambda_2 > 0$ are regularisation factors, and $\|\cdot\|_2$ denotes Frobenius norm.

At test time, for each user we are given an unseen set of items, the task is now to produce a ranked list. Since the parameters W and H have been learnt, we now can compute the value function for every user-item pair using $f(i,u)$ in Eq. (22), which can then be used to sort the items.

5 Evaluation

In this section we present evaluation results of our proposed PMOP on two tasks: document ranking on Web data and collaborative filtering on movie data. We implement three methods resulting from our framework (see description in Section 4.1). The first is the PMOP with arithmetic mean SUFs (denoted by PMOP-AS), the second is with Gibbs sampling (denoted by PMOP-Gibbs), and the third is with Metropolis-Hastings sampling (denoted by PMOP-MH).

Two performance metrics are reported: the Normalised Discounted Cumulative Gain at position T (NDCG@T) [30], and the Expected Reciprocal Rank (ERR) [9]. NDCG@T metric is defined as

$$\text{NDCG@}T \quad = \frac{1}{\kappa(T)} \quad \sum_{i=1}^{T} \frac{2^{r_i} - 1}{\log_2(1+i)}$$

where r_i is the relevance judgment of the object at position i, $\kappa(T)$ is a normalisation constant to make sure that the gain is 1 if the ordering is correct. The ERR is defined as

$$\text{ERR} \quad = \quad \sum_i \frac{1}{i} V(r_i) \prod_{j=1}^{i-1} (1 - V(r_j)) \quad \text{where } V(r) = \frac{2^r - 1}{2^{r_{max}}}.$$

Both the metrics emphasize the importance of the top-ranked objects, which is essential for finding just a few relevant objects from a large collection.

5.1 Web Document Ranking

5.1.1 Data and Settings

We employed the dataset from the recent Yahoo! learning-to-rank challenge [8]. The data contains the groundtruth relevance labels for 19,944 queries with totally 473,134 documents. The labels are numerical scores from 0 to 4 indicating the relevancy of a document with respect to a particular query. Since on average, there are 24 documents per query, the occurrences of ties are frequent. As we are mainly concerned with modelling preferences, we use the features for each document-query pairs supplied by Yahoo!, and there are 519 unique features.

We split the data into two sets: the training set contains roughly 90% queries (18,425 queries, 471,614 documents), and the test set is the remaining 10% (1,520 queries, 47,313 documents). We then normalise the features across the whole training set to have mean 0 and standard deviation 1. For comparison, we implement several well-known methods, including RankNet [6], linear Ranking SVM [31] and ListMLE [52]. The RankNet and Ranking SVM are pairwise methods, and they differ on the choice of loss functions, i.e. logistic loss for the RankNet and hinge loss for the Ranking SVM[20]. Similarly, choosing quadratic loss gives us a rank regression method, which we will call Rank Regress (see Appendix A.2 for more details).

We also implement two variants of the Bradley-Terry model with ties handling, one by Rao-Kupper [43] (denoted by PairTies-RK; this also appears to be implemented in [54] under the functional gradient setting) and another by Davidson [13] (denoted by PairTies-D; and this is the first time the Davidson's method is applied to learning-to-rank). See Appendix A.3 for implementation details.

For those pairwise methods without ties handling, we simply ignore the tied document pairs. For the ListMLE, we simply sort the documents within a query by relevance scores, and those with ties are ordered according to the sorting algorithm. All methods, except for PMOP-Gibbs/MH, are trained using the Limited Memory Newton Method known as L-BFGS. The L-BFGS is stopped if the relative improvement over the loss is less than 10^{-5} or after 100 iterations. As the PMOP-Gibbs/MH are stochastic, we run the MCMC for a few steps per query, then update the parameter using the Stochastic Gradient Ascent. The learning rate is fixed to 0.1, and the learning is stopped after 1,000 iterations.

[20] Strictly speaking, RankNet makes use of neural networks as the scoring function, but the overall loss is still logistic, and for simplicity, we use simple perceptron.

	First-order features			Second-order features		
	ERR	NDCG@1	NDCG@5	ERR	NDCG@1	NDCG@5
Rank Regress	0.4882	0.683	0.6672	0.4971	0.7021	0.6752
RankNet	0.4919	0.6903	0.6698	0.5049	0.7183	0.6836
Ranking SVM	0.4868	0.6797	0.6662	0.4970	0.7009	0.6733
ListMLE	0.4955	0.6993	0.6705	0.5030	0.7172	0.6810
PairTies-D	0.4941	0.6944	0.6725	0.5013	0.7131	0.6786
PairTies-RK	0.4946	0.6970	0.6716	0.5030	0.7136	0.6793
PMOP-AS	0.5038	0.7137	0.6762	0.5086	**0.7272**	**0.6858**
PMOP-AS(*)	**0.5054**	0.7115	0.6763	**0.5099**	0.7243	0.6835
PMOP-Gibbs	0.5037	0.7105	**0.6792**	0.5040	0.7124	0.6706
PMOP-MH	0.5045	**0.7139**	0.6790	0.5053	0.7122	0.6713

Table 2: Performance measured in ERR and NDCG@T. PairTies-D and PairTies-RK are the Davidson's method and Rao-Kupper's method for ties handling, respectively. PMOP-AS is the PMOP with arithmetic mean SUFs, where PMOP-AS(*) means likelihood weighting in Eq. (18) has been used. PMOP-Gibbs/MH are the PMOP with Gibbs/Metropolis-Hasting sampling, respectively (using geometric mean SUFs). See Section 4.1 for detailed description.

We also implement enhancements described in Sections 3.5 and 4.1.3. In particular, for the weight sequence in Equation 18, we use $\alpha_k = 1 + \log r_k$ where r_k is the relevance label of the k-th subset in the training data. First-order features are precomputed by Yahoo! and second-order features are constructed based on first-order features. The selection threshold ρ is set at 0.15 yielding $14,425$ second-order features as it balances well between speed and performance.

5.1.2 Results

The results are reported in Table 2. It can be seen that modelling ties are beneficial, as PairTies-D and PairTies-RK perform better than the RankNet (which is essentially Bradley-Terry model without ties handling), and our PMOP variants improve over the ListMLE (which is a special case of PMOP without ties), despite the simplicity in the potential function choices in Eqs (20) and (21). The PMOP-MH wins over the best performing baseline, ListMLE, by 2% according to the ERR metric. In our view, this is a significant improvement given the scope of the dataset. We note that the difference in the top 20 of the leaderboard[21] of the Yahoo! challenge was just 1.56%. Overall, second-order features consistently help to improve performance of all models.

As for training time, the PMOP-AS is numerically the fastest method. Theoretically, it has the linear complexity similar to ListMLE. All other pairwise methods are quadratic in query size, and thus numerically slower (Table 1). The PMOP-Gibbs/MH is also linear in the query size, by a constant factor determined by the number of sampling iterations.

[21] Our result using second-order features was submitted to the Yahoo! challenge and obtained a position in the top 4% over 1055 teams, given that our main purpose was to propose a new theoretical and useful model.

	ERR	NDCG@1	NDCG@5
CoFi^{RANK}.Regress	0.683	0.574	0.592
CoFi^{RANK}.Ordinal	0.631	0.504	0.528
CoFi^{RANK}.NDCG@10	0.598	0.471	0.487
PMOP-AS	**0.684**	**0.576**	**0.602**

Table 3: Results for movie recommendation.

5.2 Collaborative Filtering for Movie Recommendation

5.2.1 Data and Settings

We use the MovieLens $10M$ data[22], which has slightly over 10 million ratings applied to $10,681$ movies by $71,567$ users. The ratings are from 0.5 to 5 with 0.5 increments. To facilitate ties, we round ratings to the nearest integers. In this setting, the user and their rating history play the role of context, and the movies the role of objects. However, different from the document ranking setting, the user tastes need to be discovered from the data rather than being given. This would mean that a recommendation for a particular user relies on the information captured from the preferences expressed by the other users.

One property of movie data is that there are movies whose quality is inherently bad or good, and thus the audience generally agree about their ratings. To make the data more challenging, we first remove about half of the movies with low diversity in rating among all users. To estimate the rating diversity, we use the entropy measure for each movie i: $-\sum_{r=1}^{r_{max}} \hat{P}(r) \log \hat{P}(r)$ where $r_{max} = 5$ is the rating scale, and $\hat{P}(r)$ is the portion of users who have rated the movie by r points. The low entropy means that the distribution $\hat{P}(r)$ is peaked around a particular rating, or equivalently the less diversity among users. On the other hand, the high entropy indicates that the distribution is closer to uniform, or equivalently minimum agreement among users. Further, for each user, we randomly select 50 movies for training, and the rest for testing. To ensure that there are at least 10 test movies for each user, we remove those users with less than 60 ratings. This leaves us $21,649$ users on $5,247$ movies.

For comparison we run the CoFi^{RANK}-NDCG algorithm of [51] on the data with the code provided by the authors[23]. In Eq. (23), matrices W (user-based parameters) and H (movie-based parameters) are initialised randomly in the range $[0, \frac{1}{2}\sqrt{r_{max}/D}]$ where D is the hidden dimensionality, and $\beta = 0.01$. For this problem, we use only the PMOP-AS for efficiency, where parameters are learnt using gradient ascent. For all algorithms, we set the feature dimensionality to $D = 100$.

5.2.2 Results

We first study the ability to fit the data of PMOP-AS (with tie handling) compared with the standard Plackett-Luce (without tie handling) in terms of data likelihood. For the remaining movies for each user, we randomly picked $M \in \{5, 10, 20, 30, 40, 50\}$ movies for model fitting

[22] http://grouplens.org/node/73

[23] The code is available at: http://www.cofirank.org/downloads. We implemented a simple wrapper to compute the ERR and NDCG scores (at various positions), which are not available in the code.

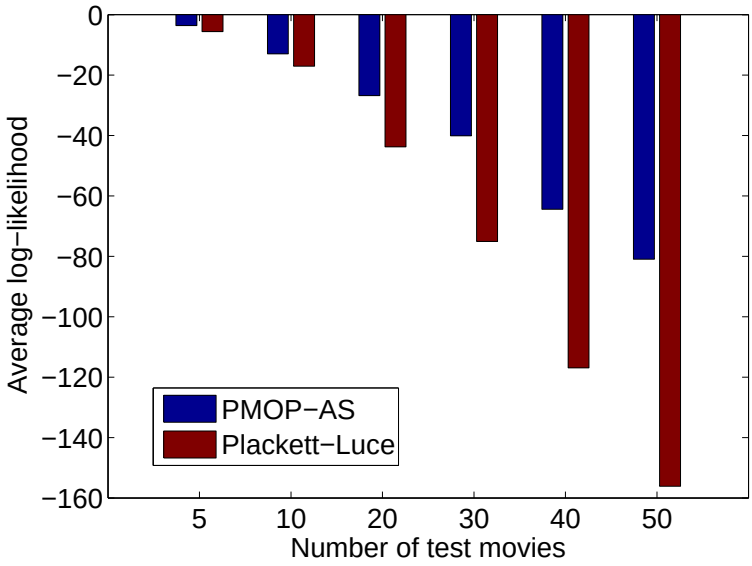

Figure 2: Log-likelihood on test movies. The ordering for Plackett-Luce is based on a sort algorithm, where ties may be broken arbitrarily. The higher likelihood, the better fitting.

evaluation (those whose number of remaining movies is less than M are not included). Figure 2 shows that the PMOP-AS fits the test data better than the Plackett-Luce. The difference is more dramatic when M is large. This is expected since the Plackett-Luce does not account for the ties, which occur more frequently with larger numbers of movies.

We then compare the predictive performance of the PMOP-AS for the recommendation of unseen movies to users. Table 3 reports results against three variants of the Cofi^{RANK}, which is one of the best-known algorithms in this class of problems. It can be seen that our PMOP-AS is competitive as it is as good as or better than the Cofi^{RANK}.

6 Discussion

In our specific choice of the local distribution in Eq (8), we share the same idea with that of Plackett-Luce, in which the probability of choosing a subset is proportional to its utility. In fact, when we limit the subset size to 1, i.e. there are no ties, the proposed model reduces to the well-known Plackett-Luce models.

The distribution of the case of arithmetic mean SUFs has an interesting interpretation. From Eq (13) the local partition distribution can be rewritten as

$$P\left(X_k \mid X_{1:k-1}\right) = \frac{1}{C\left|X_k\right|} \sum_{x \in X_k} \frac{\phi_k(x)}{\sum_{x' \in R_k} \phi_k(x')}$$

Since $\phi_k(x) / \sum_{x' \in R_k} \phi_k(x')$ is the probability of choosing x as the top object at stage k, $P\left(X_k \mid X_{1:k-1}\right)$ can be interpreted as the probability of choosing any member in the subset X_k as the top object, up to a multiplicative constant. Thus, the this model offers a simple way to model the inherent uncertainty in the choices when ties occur. This bears some similarity

with the multiple-instance learning setting [15], where we know that at least one of the objects in the subset must have a given label (e.g. being chosen in our case).

It is worth mentioning that the factorisation in Eq (7) and the choice of local distribution in Eq (8) are not unique. In fact, the chain-rule can be applied to any sequence of choices. For example, we can factorise in a *backward* manner

$$P\left(X_1, \ldots, X_{K_\sigma}\right) = P\left(X_{K_\sigma}\right) \prod_{k=1}^{K_\sigma - 1} P\left(X_k \mid X_{k+1:K_\sigma}\right) \tag{24}$$

where $X_{k+1:K_\sigma}$ is a shorthand for $\{X_{k+1}, X_{k+2}, ..., X_{K_\sigma}\}$. Interestingly, we can interpret this reverse process as *subset elimination*: First we choose to eliminate the worst subset, then the second worst, and so on. This line of reasoning has been discussed in [18] but it is limited to 1-element subsets. However, if we are free to choose the parameterisation of $P\left(X_k \mid X_{k+1:K_\sigma}\right)$ as we have done for $P\left(X_k \mid X_{1:k-1}\right)$ in Eq (8), there is no guarantee that the forward and backward factorisations admit the same distribution.

Our model can be placed into the framework of probabilistic graphical models (e.g. see [33]). Recall that in standard probabilistic graphical models, we have a set of variables, each of which receives values from a fixed set of states. Generally, variables and states are orthogonal concepts, and the *state space* of a variable do not explicitly depends on the states of other variables[24]. In our setting, the objects play the role of the variables, and their memberships in the subsets are their states. However, since there are exponentially many subsets, enumerating the state spaces as in standard graphical models is not possible. Instead, we can consider the ranks of the subsets in the list as the states, since the ranks only range from 1 to N. Different from the standard graphical models, the variables and the states are not always independent, e.g. when the subset sizes are limited to 1, then the state assignments of variables are mutually exclusive, since for each position, there is only one object. Probabilistic graphical models are generally directed (such as Bayesian networks) or undirected (such as Markov random fields), and our PMOP can be thought of as a directed model. The undirected setting is also of great interest, but it is beyond the scope of this paper.

The main focus of this paper is modelling ties, and thus we explicitly require that (i) members of a partition are equivalent in value, and (ii) for any two partitions the ordering between them $X_{k_1} \prec X_{k_2}$ would mean that any member of X_{k_1} is preferred to all members of X_{k_2} (see Eq (4)). This appears to be rather restrictive in many applications. For example, a shopping basket may contain some low quality items but overall, the whole basket is highly preferred because it meets a family's nutrition needs within a budget constraint. Fortunately, our probabilistic modelling in Eq (8) is still applicable to these settings where preferences over sets [2][4] can be expressed as soft constraints through the SUF $\Phi_k\left(X_k\right)$.

Although we have concentrated on efficient learning of the PMOP, in many situations we may be interested in inference tasks such as predicting the optimal partition at each stage, or predicting the optimal partitioning and ordering for the whole set. The former task can be considered as finding the optimal labelling scheme in a binary Markov random field [21] along the line of reasoning with the Gibbs sampling in Section 3.3.2. The latter is however much more involved due to the hyper-exponential stage-space. Generic methods include greedy local search for local solutions or Simulated Annealing for near global solutions. It remains

[24]Note that, this is different from saying the states of variables are independent.

to define meaningful *local moves* to explore this combinatorial state-space. One candidate would be the dual *split/merge* operator: *split* a partition into two successive sub-partitions, and *merge* two successive partitions into one.

7 Conclusions

In this paper, we have addressed the problem of modelling and learning context-specific preferences with ties. We first cast the problem into a more generic form of set partitioning and ordering. This resulted in a hyper-exponential state-space which grows as quickly as $N!/(2\,(\ln 2)^{N+1})$ for a set of N objects. Addressing this complexity, we proposed a stagewise approach in which a partition is chosen at each stage, thus reducing the state-space significantly to $\left(2^{N_k}-1\right)$ where N_k is the number of remaining objects after $(k-1)$ steps. Inference in this reduced space can then be performed using MCMC techniques - we offered a Gibbs sampling and a Metropolis-Hastings procedure. We demonstrated that with these techniques, we can proceed to train powerful models on hundreds of thousands of objects. We also proved that there exists a highly interpretable case where learning can be carried out in *linear* time. We evaluated the proposed models on two problems: the first is Web document ranking with the large-scale data from the recent Yahoo! challenge and the second is collaborative filtering with the MovieLens $10M$ dataset. The experimental results demonstrated that our proposed models are competitive against state-of-the-arts which are designed specifically for the problems.

There is room for future advancement of the current work. Firstly, we have relied on explicit expression of preferences and trained the models in a supervised manner. Discovering implicit preferences would enhance our understanding of users and offer better personalised experience. Secondly, we have assumed that the preference models remain constant over time. This assumption may not hold in the real-world but adapting to preference change is required. There is also a wide range of important applications that fit into the framework we have just presented, for example, multimedia retrieval, answer rankings and advertisements placement to name a few.

A Additional Results

A.1 Computing C

Let us calculate the constant C in Eq. (12) by first rewriting this equation for ease of comprehension

$$\sum_{S\in 2^{R_k}} \frac{1}{|S|} \sum_{x\in S} \phi_k\left(x\right) = C \times \sum_{x\in R_k} \phi_k(x)$$

where 2^{R_k} is the power set of R_k, or the set of all non-empty subsets of R_k. Equivalently

$$C = \sum_{S\in 2^{R_k}} \frac{1}{|S|} \sum_{x\in S} \frac{\phi_k\left(x\right)}{\sum_{x\in R_k} \phi_k(x)}$$

If all objects are the same, then this can be simplified to

$$C = \sum_{S \in 2^{R_k}} \frac{1}{|S|} \sum_{x \in S} \frac{1}{N_k} = \frac{1}{N_k} \sum_{S \in 2^{R_k}} 1 = \frac{2^{N_k} - 1}{N_k}$$

where $N_k = |R_k|$. In the last equation, we have made use of the fact that $\sum_{S \in 2^{R_k}} 1$ is the number of all possible non-empty subsets, or equivalently, the size of the power set, which is known to be $2^{N_k} - 1$. One way to derive this result is to imagine a collection of N_k variables, each has two states: `selected` and `notselected`, where `selected` means that the object belongs to a subset. Since there are 2^{N_k} such configurations over all states, the number of non-empty subsets must be $2^{N_k} - 1$.

For arbitrary objects, let us examine the the probability that object x belongs to a subset of size m, which is $\frac{m}{N_k}$. Recall from standard combinatorics that the number of m-element subsets is the binomial coefficient $\binom{N_k}{m}$, where $1 \leq m \leq N_k$. Thus the number of times an object appears in any m-subset is $\binom{N_k}{m} \frac{m}{N_k}$. Taking into account that this number is weighted down by m (i.e. $|S|$ in Eq. (12)), the contribution towards C is then $\binom{N_k}{m} \frac{1}{N_k}$. Finally, we can compute the constant C, which is the weighted number of times an object can belong to any subset of any size, as follows

$$C = \sum_{m=1}^{N_k} \binom{N_k}{m} \frac{1}{N_k} = \frac{1}{N_k} \sum_{m=1}^{N_k} \binom{N_k}{m} = \frac{2^{N_k} - 1}{N_k}$$

We have made use of the known identity $\sum_{m=1}^{N_k} \binom{N_k}{m} = 2^{N_k} - 1$.

A.2 Pairwise Losses

Let $f(x_i, w)$ be the scoring function parameterised by w that takes the input vector x_i and outputs a real value indicating the utility of object i. Let $\delta_{ij}(w) = f(x_i, w) - f(x_j, w)$. Pairwise models are quite similar in their general setting. The only difference is the specific loss function:

$$\ell(x_i \succ x_j; w) = \begin{cases} \log(1 + \exp\{-\delta_{ij}(w)\}) & \text{(RankNet)} \\ \max\{0, 1 - \delta_{ij}(w)\} & \text{(Ranking SVM)} \\ (1 - \delta_{ij}(w))^2 & \text{(Rank Regress)} \\ \exp\{-\delta_{ij}(w)\} & \text{(RankBoost)} \end{cases}$$

However, these losses behave quite differently from each other. For the RankNet and RankBoost, minimising the loss would widen the margin between the score for x_i and x_j as much as possible. The difference is that the RankNet is less sensitive to noise due to the log-scale. The Ranking SVM, however, aims just about to achieve the margin of 1, and the Rank Regress, attempts to bound the margin by 1.

At first impression, the cost for gradient evaluation in pairwise losses would be $\mathcal{O}(0.5N(N-1)F)$ where F is the number of parameters. However, we can achieve $\max\{\mathcal{O}(0.5N(N -$

1)), $\mathcal{O}(NF)\}$ as follows. The overall loss for a particular query is

$$\mathfrak{L} = \sum_{i,j|x_i \succ x_j} \ell(x_i \succ x_j; w)$$

Taking derivative with respect to w yields

$$\begin{aligned}\frac{\partial \mathfrak{L}}{\partial w} &= \sum_{i,j|x_i \succ x_j} \frac{\partial \ell(x_i \succ x_j; w)}{\partial \delta_{ij}} \left(-\frac{\partial f_i}{\partial w} + \frac{\partial f_j}{\partial w}\right) \\ &= -\sum_i \frac{\partial f_i}{\partial w} \sum_{j|x_i \succ x_j} \frac{\partial \ell(x_i \succ x_j; w)}{\partial \delta_{ij}} + \sum_j \frac{\partial f_j}{\partial w} \sum_{i|x_i \succ x_j} \frac{\partial \ell(x_i \succ x_j; w)}{\partial \delta_{ij}}\end{aligned}$$

As $\left\{\frac{\partial \ell(x_i \succ x_j; w)}{\partial \delta_{ij}}\right\}_{i,j|x_i \succ x_j}$ can be computed in $\mathcal{O}(0.5N(N-1))$ time, and $\left\{\frac{\partial f_i}{\partial w}\right\}_i$ in $\mathcal{O}(NF)$ time, the overall cost would be $\max\{\mathcal{O}(0.5N(N-1)), \mathcal{O}(NF)\}$.

A.3 Learning the Pairwise Models with Ties

This subsection describes the details of learning the paired ties models discussed in Section 6.

A.3.1 Davidson's Method

Recall from Section 2 that in the Davidson's method the probability masses are defined as

$$P(x_i \succ x_j; w) = \frac{1}{Z_{ij}}\phi(x_i); \qquad P(x_i \sim x_j; w) = \frac{1}{Z_{ij}}\nu\sqrt{\phi(x_i)\phi(x_j)}$$

where $Z_{ij} = \phi(x_i) + \phi(x_j) + \nu\sqrt{\phi(x_i)\phi(x_j)}$ and $\nu \geq 0$. For the simplicity of unconstrained optimisation, let $\nu = e^{\beta}$ for $\beta \in \mathbb{R}$. Let $P_i = P(x_i \succ x_j; w)$, $P_j = P(x_j \succ x_i; w)$ and $P_{ij} = P(x_i \sim x_j; w)$.

Taking derivatives of the log-likelihood gives

$$\begin{aligned}\frac{\partial \log P(x_i \succ x_j; w)}{\partial w} &= (P_j + 0.5P_{ij})\frac{\partial \log \phi(x_i, w)}{\partial w} - (P_i + 0.5P_{ij})\frac{\partial \log \phi(x_j, w)}{\partial w} \\ \frac{\partial \log P(x_i \succ x_j; w)}{\partial \beta} &= -P_{ij} \\ \frac{\partial \log P(x_i \sim x_j; w)}{\partial w} &= (0.5 - P_i - 0.5P_{ij})\frac{\partial \log \phi(x_i, w)}{\partial w} \\ &\quad +(0.5 - P_j - 0.5P_{ij})\frac{\partial \log \phi(x_j, w)}{\partial w} \\ \frac{\partial \log P(x_i \sim x_j; w)}{\partial \beta} &= 1 - P_{ij}.\end{aligned}$$

A.3.2 Rao-Kupper's Method

Recall from Section 2 that the Rao-Kupper's model defines the following probability masses

$$\begin{aligned}P(x_i \succ x_j; w) &= \frac{\phi(x_i)}{\phi(x_i) + \theta\phi(x_j)} \\ P(x_i \sim x_j; w) &= \frac{(\theta^2 - 1)\phi(x_i)\phi(x_j)}{[\phi(x_i) + \theta\phi(x_j)]\,[\theta\phi(x_i) + \phi(x_j)]}\end{aligned}$$

where $\theta \geq 1$ is the ties factor and w is the model parameter. Note that $\phi(.)$ is also a function of w, which we omit here for clarity. For ease of unconstrained optimisation, let $\theta = 1 + e^{\alpha}$ for $\alpha \in \mathbb{R}$. In learning, we want to estimate both α and w. Let

$$\begin{aligned} P_i &= \frac{\phi(x_i)}{\phi(x_i) + (1+e^{\alpha})\phi(x_j)}; \qquad P_j^* = \frac{\phi(x_j)}{\phi(x_i) + (1+e^{\alpha})\phi(x_j)}; \\ P_i^* &= \frac{\phi(x_i)}{(1+e^{\alpha})\phi(x_i) + \phi(x_j)}; \qquad P_j = \frac{\phi(x_j)}{(1+e^{\alpha})\phi(x_i) + \phi(x_j)}. \end{aligned}$$

Taking partial derivatives of the log-likelihood gives

$$\begin{aligned} \frac{\partial \log P(x_i \succ x_j; w)}{\partial w} &= (1 - P_i)\frac{\partial \log \phi(x_i, w)}{\partial w} - (1+e^{\alpha})P_j \frac{\partial \log \phi(x_j, w)}{\partial w} \\ \frac{\partial \log P(x_i \succ x_j; w)}{\partial \alpha} &= -P_j e^{\alpha} \\ \frac{\partial \log P(x_i \sim x_j; w)}{\partial w} &= (1 - P_i - (1+e^{\alpha})P_i^*)\frac{\partial \log \phi(x_i, w)}{\partial w} \\ &+ (1 - P_j - (1+e^{\alpha})P_j^*)\frac{\partial \log \phi(x_j, w)}{\partial w} \\ \frac{\partial \log P(x_i \sim x_j; w)}{\partial \alpha} &= \left(\frac{2(1+e^{\alpha})}{(1+e^{\alpha})^2 - 1} - P_i^* - P_j^*\right) e^{\alpha}. \end{aligned}$$

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] M. Binshtok, R.I. Brafman, C. Domshlak, and S.E. Shimony. Generic preferences over subsets of structured objects. *Journal of Artificial Intelligence Research*, 34(1):133–164, 2009.

[3] R.A. Bradley and M.E. Terry. Rank analysis of incomplete block designs. *Biometrika*, 39:324–345, 1952.

[4] G. Brewka and S. Woltran. Representing preferences among sets. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 2010.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[6] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proc. of ICML*, page 96, 2005.

[7] 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*, page 136. ACM, 2007.

[8] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *JMLR Workshop and Conference Proceedings*, volume 14, pages 1–24, 2011.

[9] O. Chapelle, D. Metlzer, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *CIKM*, pages 621–630. ACM, 2009.

[10] H. Chen, SRK Branavan, R. Barzilay, D.R. Karger, et al. Content modeling using latent permutations. *JAIR*, 36:129–163, 2009.

[11] W. Chu and S.S. Keerthi. Support vector ordinal regression. *Neural computation*, 19(3):792–815, 2007.

[12] W.W. Cohen, R.E. Schapire, and Y. Singer. Learning to order things. *J Artif Intell Res*, 10:243–270, 1999.

[13] R.R. Davidson. On extending the Bradley-Terry model to accommodate ties in paired comparison experiments. *Journal of the American Statistical Association*, 65(329):317–328, 1970.

[14] P. Diaconis. *Group representations in probability and statistics*. Institute of Mathematical Statistics Hayward, CA, 1988.

[15] T.G. Dietterich, R.H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[16] C. Domshlak, E. Hüllermeier, S. Kaci, and H. Prade. Preferences in AI: An overview. *Artificial Intelligence*, 175(7-8):1037–1052, 2011.

[17] J. Doyle. Prospects for preferences. *Computational Intelligence*, 20(2):111–136, 2004.

[18] M.A. Fligner and J.S. Verducci. Multistage ranking models. *Journal of the American Statistical Association*, 83(403):892–901, 1988.

[19] Y. Freund, R. Iyer, R.E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4(6):933–969, 2004.

[20] J. Fürnkranz and E. Hüllermeier. *Preference learning*. Springer-Verlag New York Inc, 2010.

[21] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 6(6):721–742, 1984.

[22] W.A. Glenn and H.A. David. Ties in paired-comparison experiments using a modified Thurstone-Mosteller model. *Biometrics*, 16(1):86–109, 1960.

[23] Y. Guo and C. Gomes. Learning optimal subsets with implicit user preferences. In *Proceedings of the 21st international joint conference on Artifical intelligence*, 2009.

[24] V. Ha and P. Haddawy. Similarity of personal preferences: Theoretical foundations and empirical analysis. *Artificial Intelligence*, 146(2):149–173, 2003.

[25] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[26] G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[27] J. Huang, C. Guestrin, and L. Guibas. Fourier theoretic probabilistic inference over permutations. *The Journal of Machine Learning Research*, 10:997–1070, 2009.

[28] T.K. Huang, R.C. Weng, and C.J. Lin. Generalized Bradley-Terry models and multi-class probability estimates. *The Journal of Machine Learning Research*, 7:115, 2006.

[29] E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 2008.

[30] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):446, 2002.

[31] T. Joachims. Optimizing search engines using clickthrough data. In *Proc. of SIGKDD*, pages 133–142. ACM New York, NY, USA, 2002.

[32] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, 2008.

[33] S.L. Lauritzen. *Graphical Models*. Oxford Science Publications, 1996.

[34] N.N. Liu, M. Zhao, and Q. Yang. Probabilistic latent preference analysis for collaborative filtering. In *CIKM*, pages 759–766. ACM, 2009.

[35] T.Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[36] T. Lu and C. Boutilier. Learning mallows models with pairwise preferences. *ICML-11, Bellevue, WA*, 2011.

[37] R.D. Luce. *Individual choice behavior*. Wiley New York, 1959.

[38] C.L. Mallows. Non-null ranking models. I. *Biometrika*, 44(1):114–130, 1957.

[39] J.I. Marden. *Analyzing and modeling rank data*. Chapman & Hall/CRC, 1995.

[40] M. Mureşan. *A concrete approach to classical analysis*. Springer Verlag, 2008.

[41] R.M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

[42] R.L. Plackett. The analysis of permutations. *Applied Statistics*, pages 193–202, 1975.

[43] P.V. Rao and L.L. Kupper. Ties in paired-comparison experiments: A generalization of the Bradley-Terry model. *Journal of the American Statistical Association*, pages 194–204, 1967.

[44] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994. ACM.

[45] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM Press New York, NY, USA, 2001.

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

[47] T. Truyen, D.Q Phung, and S. Venkatesh. Probabilistic models over ordered partitions with applications in document ranking and collaborative filtering. In *Proc. of SIAM Conference on Data Mining (SDM)*, Mesa, Arizona, USA, 2011. SIAM.

[48] T.T. Truyen, D.Q. Phung, and S. Venkatesh. Preference networks: Probabilistic models for recommendation systems. In P. Christen, P.J. Kennedy, J. Li, I. Kolyshkina, and G.J. Williams, editors, *The 6th Australasian Data Mining Conference (AusDM)*, volume 70 of *CRPIT*, pages 195–202, Gold Coast, Australia, Dec 2007. ACS.

[49] J.H. van Lint and R.M. Wilson. *A course in combinatorics*. Cambridge Univ Pr, 1992.

[50] M.N. Volkovs and R.S. Zemel. BoltzRank: learning to maximize expected ranking gain. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM New York, NY, USA, 2009.

[51] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. CoFi^{RANK}-maximum margin matrix factorization for collaborative ranking. *Advances in neural information processing systems*, 20:1593–1600, 2008.

[52] F. Xia, T.Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *Proc. of ICML*, pages 1192–1199, 2008.

[53] L. Younes. Parametric inference for imperfectly observed Gibbsian fields. *Probability Theory and Related Fields*, 82(4):625–645, 1989.

[54] K. Zhou, G.R. Xue, H. Zha, and Y. Yu. Learning to rank with ties. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–282. ACM, 2008.