Under consideration for publication in Knowledge and Information Systems

Modelling Human Preferences for Ranking and Collaborative Filtering: A Probabilistic Ordered Partition Approach

Truyen Tran[a,b], Dinh Phung[a] and Svetha Venkatesh[a]

[a]Center for Pattern Recognition and Data Analytics, Deakin University, Australia.
[b]Department of Computing, Curtin University, Australia.
Emails: {truyen.tran, dinh.phung, svetha.venkatesh}@deakin.edu.au

Abstract. Learning preference models from human generated data is an important task in modern information processing systems. Its popular setting consists of simple input ratings, assigned with numerical values to indicate their relevancy with respect to a *specific* query. Since ratings are often specified within a small range, several objects may have the same ratings, thus creating *ties* among objects for a given query. Dealing with this phenomena presents a general problem of modelling *preferences* in the presence of *ties* and being *query-specific*. To this end, we present in this paper a novel approach by constructing probabilistic models directly on the collection of objects 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 order among them. Learning and inference in such a large state space are challenging, and yet we present in this paper efficient algorithms to perform these tasks. Our approach exploits discrete choice theory, imposing generative process such that the 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 presented for the proposed models. We demonstrate that the model can potentially be trained in a large-scale setting of hundreds of thousands objects using an ordinary computer. In fact, in some special cases with appropriate model specification, our models can be learned in *linear* time. We evaluate the models on two application areas: (i) document ranking with the data from the Yahoo! challenge, and (ii) collaborative filtering with movie data. We demonstrate that the models are competitive against state-of-the-arts.

Received xxx
Revised xxx
Accepted xxx

1. Introduction

Modelling user preferences is key to success of many modern applications in Web search, recommended systems and social media. One important task is to rank a set of objects in a decreasing order of preference with respect to a query submitted by an user. Objects can be anything of interest (e.g. Web documents, images, shopping items, friends or banking services) whilst queries can be anything that triggers the object response (e.g. keywords, user purchasing history or user profiles). Handcrafting a good preference model is likely a complex and expensive task which involves a great amount of domain knowledge. It is therefore desirable that preference models can be automatically learnt from preference data which can be acquired from ordinary users 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 query. 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 query.

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 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 users' preferences with *ties* at query-level. The answer to this question is the focus of this paper.

Previous work on tied preferences is fairly limited and typically considers pairwise comparisons [13][20][42][55]. Little work has considered modelling at query-level preferences. We take an alternative approach by *modelling objects with the same tie with respect to the same query as a partition, hence translating the original problem into ordering these partitions instead*. This new formulation results in a combinatorial problem which involves 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 the most general and theoretically adequate, we need to explore in a super-exponential combinatorial state-space with unknown numbers of partitions and unknown ordering among them. To be more precise, the size of the state-space of N objects grows exponentially as $N!/(2\,(\ln 2)^{N+1})$ [39, pp. 396–397]. Learning and inference in such 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 query-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 partition can then be composed using a variant of the Plackett-Luce model [35][41] but at the level of partitions rather than objects. This approach offers two important benefits: (i) theoretically, it is interpretable in modelling user choices [35] in that users choose an object with probability proportional to its *worth*; and (ii) practically, this greatly reduces the size of state-space

[1] This rating-to-rank conversion is not reversible since we cannot generally infer ratings from a ranking. First, the top rating for each query 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.

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 computer. Another surprising result is that, with appropriate choice of subset-wise worth, we can learn the models in *linear* time. This is a great saving, even is more efficient than existing pairwise models, where the time complexity is generally quadratic in query sizes. We specify this model as the *Probabilistic Model over Ordered Partitions* (PMOP).

We apply the proposed model for two problems. The first 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 need, 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 [5], (linear) Ranking SVM [27] and ListMLE [53].

We further present an application of the model for movie recommendation, where each user has expressed their preferences in term of ratings on a small number of movies, and the task is to recommend new movies they might 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 is an implicit query, 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 [52].

Our contribution, to the best our our knowledge, is the first to address the problem of preference modelling 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 functions, yielding learning algorithms 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 [16][35][41]. The significance of the model is its potential for use in many recommender applications, noticeably in ranking and collaborative filtering tasks. 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 the two applications of the proposed. Further theoretical implication is discussed in Section 6. Finally, Section 7 concludes the paper.

2. Background

In this section, we present necessary background related to our problem of preference modeling with ties, Web document ranking and collaborative filtering.

Preference Modelling. The notion of preference is widely studied in many areas,

including economics, statistics, artificial intelligence, machine learning and data mining (e.g., see [18][37]). A typical setting is a collection of observed pairwise preferences, each indicates an object x_i is preferred to another object x_j, denoted by $x_i \succ x_j$. Statistical approach often involves estimating the probability that a particular preference order occurs, i.e. $P(x_i \succ x_j)$. One popular choice 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 worth (or importance) or an object. This expression asserts that the probability of choosing an object over another is proportional to its worth. Later, this line of reasoning was generalised into the theory of discrete choice by Luce [35]. 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 an complete ordering $x_1 \succ x_2 \succ ... \succ x_N$

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

This model was first studied by Luce in [35], subsequently by Plackett in [41], and hence we shall refer to as the Plackett-Luce model. The idea of this model is that, we proceed to 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 [31][36], 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 and 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), and $\tau(.,.)$ is a rank-based distance function. Depending on the distance measures, the model may differ; and the popular distance measures include those by Kendall [28] and Spearman [46]. However, there are several drawbacks of this approach. First, assuming a single mode over all possible rankings is rather restrictive since in many application areas involving users, there should be at least a mode per user cluster. 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 which treats an ordering as an element of a symmetric group in the group theory [14][24]. However, since inference is complex, there has not been any sizable applications in practice following this approach.

Query-Level Preferences and Tie Modelling. We are interested in the setting that each query is associated with a subset of objects which are more or less related to it. For example, the query can be a set of keywords which triggers responses from a typical search engine. Less explicitly, it can be an user with a rating history whose will implicitly ask for a recommended item list from a shopping site. In machine learning under the setting of *label ranking* (e.g. see [50] for a recent survey), on the other hand, content would play the role of a query and labels the objects to be ranked. Usually

the subset is smaller than the original set, 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. 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 query. As a result, we may have multiple preference models, one per query, sharing the same parameters.

Preference with ties has also been studied for several decades, mostly in the statistical communities [13][20][42][55]. The general idea is to allocate some probability mass for the event of ties, e.g. when $P(x_i \sim x_j) \neq 0$, where $\sim$ denotes no preferences. 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 the model parameter controlling the contribution of ties. This model is quite intuitive in the sense that when tie occurs, both objects contributes equally to the probability. When $\nu = 0$, this reduces to the standard Bradley-Terry model. In a similar manner, Rao and Kupper [42] 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 [25]. However, this setting is inflexible since the the partitioning into groups is known in advance, and the groups behave just like standard super-objects. Another way to deal with ties is to create an 'equivalent permutation set' (e.g. see [34]) from ties and then train with the full-rank algorithms. The idea is to minimise the min loss over the set in a fashion similar to *multiple-instance* learning [15].

To handling ties at the query-level in the general form, our initial work has proposed a probabilistic framework over ordered partitions (PMOP) in [48]. The framework assumes no prior knowledge of partitioning and ordering. Instead it models a generative stagewise process where each partition is selected at a stage. As a result, the PMOP is a Plackett-Luce model at the set level, where sets are automatically selected from the collection of query-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 a weighting scheme to bias the data likelihood towards performance metrics, and 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. Web document ranking is the heart of any search engines. 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 document relevancy and quality. Later development exploited the hyperlink structure of the Web, from which global ranking models such as PageRank [4] are used to compute the general quality of a Web page. However, this ranking does not necessarily capture the hidden user preferences. In addition, how to combine these measures still remains an art rather than science.

A more recent trend is to learn search preference models directly from labelled data. Typically, based on the query, document content, link structure and user search history, context and profiles, a set of features is extracted. The goal is to estimate a preference function that takes the features as input and outputs a real score, from which documents are ranked. This trend is called *learning-to-rank* - a subfield of machine learning dealing with preference modelling. As discussed in [34], machine learning methods extended to ranking can be divided into:

- *Pointwise approach* which includes methods such as ordinal regression [10][12]. Each query-object pair is assigned an ordinal label, e.g. from the set $\{0, 1, 2, ..., M\}$. This simplifies the problem as we do not need to worry about the exponential number of permutations. The complexity is therefore linear in the number of query-object pairs. The drawback is that the ordering relation between objects is not explicitly modelled.
- *Pairwise approach* which spans preference to binary classification [5][11][17][27] methods, where the goal is to learn a classifier that can separate two objects (per query). This casts the ranking problem into a standard binary classification framework, wherein many algorithms are readily available, for example, SVM [27], neural network and logistic regression [5], and boosting [17]. The complexity is quadratic in number of objects per query and linear in number of queries. Again, this approach ignores the simultaneous interaction among objects within the same query.
- *Listwise approach* which models the distribution of permutations [6][51][53]. The ultimate goal is to model a full distribution of all permutations, and the prediction phase outputs the most probable permutation. This approach appears to be most natural for the ranking problem. In fact, the methods suggested in [6][53] are applications of the Plackett-Luce model.

Collaborative Filtering. Collaborative filtering [43] is an highly successful approach to recommendation [1][2][32]. Started in early 1990s, collaborative filtering has been one of the most active research topics in information systems. The applications range widely from movie recommendation, shopping item suggestion to financial advise and drug discovery. Recently, it has appeared from the Netflix challenge[2] that the two most effective methods are the neighbourhood-based (e.g. see [43][44]) and the latent traits discovery (e.g. see [29]). In neighbourhood-based method, 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 latent traits methods, we want to discover the hidden tastes of an user and profiles of an item. In particular in the matrix factorisation approach to latent traits, a rating is approximated

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

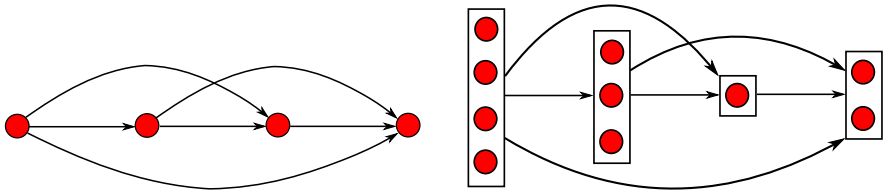

Figure 1. Complete ordering (left) versus subset ordering (right). For the subset ordering, the bounding boxes represents the subsets of elements of the same rank. Subset sizes are $4, 3, 1, 2$, respectively.

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 worth rather than rating.

One important issue with current practice in collaborative filtering is that it is largely based on rate prediction while the real goal is to rank items according to user's preferences. Although the rating could be used for ranking, it is better to directly address the rank problem in the first place [33, 45, 52]. Our work is in preference modelling, and thus deals with 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 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. In a complete ranking setting, each object x_i is further assigned with a ranking index π_i, resulting in the ranked list of $\{x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_N}\}$ where $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_N)$ is a permutation over $\{1, 2, \ldots, N\}$. For example, X might be a set of objects that are related to a query, and π_1 is the index to the first object, π_2 is the index to second object and so on. Ideally $\boldsymbol{\pi}$ should contain ordering information for all objects in the set; however, this task is not always possible for any non-trivial size N due to the labor cost involved[3]. Instead, in many situations, during training a object is *rated*[4] to indicate the its degree of relevance for the query. This creates a scenario where more than one object will be assigned to the same rating

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

[4] We caution the confusion between 'rating' and 'ranking' here. Ranking 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.

– a situation known as '*ties*' in preference modelling. When we enumerate over each object x_i and putting those with the same rating together, the set of N objects X can now be viewed as being divided into K partitions with each partition is assigned with a number to indicate the its unique rank $k \in \{1, 2, .., K\}$. The ranks are obtained by sorting ratings associated with each partition in the decreasing order.

Consider a more generic setting in which we know that objects will be rated against an ordinal value from 1 to K but do not know individual ratings. 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

$$X = \bigcup_{j=1}^{K} X_{\sigma_j} \tag{3}$$

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[5] of objects with the same rating k. These partitions are pairwise disjoint and having cardinality[6] range from 1 to N. 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 ranking setting 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

$$\sum_{k=1}^{N} \left| \begin{matrix} N \\ k \end{matrix} \right| k! = \text{Fubini}\,(N) = \sum_{j=1}^{\infty} \frac{j^N}{2^{j+1}} \tag{4}$$

which is also known in combinatorics as the Fubini's number [39, 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 [39, pp. 396–397] to approach $N!/(2\,(\ln 2)^{N+1})$ as $N \to \infty$ where we note that $\ln(2) < 1$, and thus it grows much faster than $N!$. Clearly, for unknown K this presents a very challenging problem to inference and learning. In this paper, we shall present a generic MCMC-based approach to tackle this state-space explosion in supervised learning settings, and a specific parameterisation which leads to *linear* time.

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

[5] 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 worth.

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

given in Eq. (3):

$$P(X) = P(X_{\sigma_1}, \ldots, X_{\sigma_K}) \tag{5}$$

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 ranked, result in a ranking index vector $\boldsymbol{\sigma}$. This description is generic and one can proceed in different ways to further characterise Eq. (5). We present here a generative, multistage view to this same problem so that it lends naturally to the specification of the distribution in Eq. (6): 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 remainder set $\{X \setminus X_1\}$ to be selected, 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.[7] 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 ranked partitions as

$$\begin{aligned} P(X_1, \ldots, X_{K_\sigma}) &= P(X_1) \prod_{k=2}^{K_\sigma} P(X_k \mid X_1, \ldots, X_{k-1}) \\ &= P(X_1) \prod_{k=2}^{K_\sigma} P(X_k \mid X_{1:k-1}) \end{aligned} \tag{6}$$

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. 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[8] of R, we are ready to specify each local conditional distribution in Eq. (6) as:

$$P(X_k \mid X_{1:k-1}) = \frac{1}{Z_k} \Phi_k(X_k) \tag{7}$$

where $\Phi_k(S) \in \mathbb{R}^+$ is an order-invariant[9] set function defined over a subset or partition S, and $Z_k = \sum_{S \in 2^{R_k}} \Phi_k(S)$ is the normalising constant to ensure a proper distribution. We term our model *Probabilistic Model over Ordered Partition* (*PMOP*). See Figure 1 for a graphical illustration.

Using Eqs. (7) and (6), the log-likelihood function and its gradient, without explicit

[7] This process resembles the generative process of Plackett-Luce discrete choice model [35][41], 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.

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

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

mention of the model parameter, are:

$$\mathcal{L} = \log P\left(X_1\right) + \sum_{k=2}^{K_\sigma} \log P\left(X_k \mid X_{1:k-1}\right) \tag{8}$$

$$\partial\mathcal{L} = \sum_{k=1}^{K_\sigma} \partial \log \Phi_k\left(X_k\right) - \sum_{k=1}^{K_\sigma} \left\{ \sum_{S \in 2^{R_k}} P\left(S \mid X_{1:k-1}\right) \partial \log \Phi_k\left(S\right) \right\} \tag{9}$$

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 $P\left(X_k \mid X_{1:k-1}\right)$, which depends on Z_k as in Eq. (7). Likewise, the gradient of the log-likelihood depends on model expectation $\sum_{S \in 2^{R_k}} P\left(S \mid X_{1:k-1}\right) \partial \log \Phi_k\left(S\right)$. Unfortunately, without any assumption about the form of $\Phi_k\left(.\right)$, these two quantities cannot be evaluated exactly except for trivial cases where the set X contains just few objects.

In the rest of this section, we will concentrate our effort on solving this problem. In Section 3.3, we describe specific forms of $\Phi_k\left(.\right)$ that have exact evaluation in linear time. For general cases, we present a MCMC-based sampling methods for approximate evaluation in manageable time in Section 3.4.

3.3. Special Cases: Set Functions and Their Decomposition

We first discuss some desirable properties of the set function $\Phi_k\left(.\right)$ and then present special cases that lead to very efficient computation of data likelihood and its gradient.

3.3.1. Set Functions as Aggregation

Although it is necessary to correctly model the partitioning and ordering process at training time, it is more practical to obtain a complete ranking of objects at prediction time. This is because a complete rank is more interpretable to users so that they may decide to select only a few top ranked objects. Another practical benefit is that this helps to reduce the size of the state space significantly, from Fubini $\left(N\right)$ (simultaneous partitioning and ordering) to $N!$ (ordering only). One of the most efficient way is perhaps to assign a *worth* value to each object, and then rank the objects accordingly using standard sort algorithms in $N \log N$ on average. The question is now how to estimate the object worth during the training phase. Denote by $\phi(x) \in \mathbb{R}^+$ the worth of object x, the set function arising from Eq. (7) can then be rewritten as

$$\Phi_k\left(X_k\right) = h\left\{\phi(x) \mid x \in X_k\right\}$$

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

Regularising the set size. We can impose our prior knowledge about a set size. 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 in to $\Phi_k\left(\cdot\right)$. The case of lacking specific knowledge, however, it is important to regularise its dependency on the set size. One way to achieve this is to enforce the constraint:

$$\min_{x \in X_k} \phi(x) \leq \Phi_k\left(X_k\right) \leq \max_{x \in X_k} \phi(x)$$

In fact, there are many common aggregation functions satisfying this condition, for example the following are arranged in the decreasing order (Max $\geq$ Arithmetic mean $\geq$ Geometric mean $\geq$ Harmonic mean $\geq$ Min):

– Maximum $\quad \Phi_k(X_k) = \max_{x \in X_k} \phi(x)$

– Arithmetic mean $\quad \Phi_k(X_k) = \frac{1}{|X_k|} \sum_{x \in X_k} \phi(x)$

– Geometric mean $\quad \Phi_k(X_k) = \left\{ \prod_{x \in X_k} \phi(x) \right\}^{\frac{1}{|X_k|}}$

– Harmonic mean $\quad \Phi_k(X_k) = |X_k| \left[\sum_{x \in X_k} \frac{1}{\phi(x)} \right]^{-1}$

– Minimum $\quad \Phi_k(X_k) = \min_{x \in X_k} \phi(x)$

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 five types of aggregation function listed above satisfy this condition.

3.3.2. Arithmetic Mean Aggregation

Now we focus on a specific type of aggregation function – the arithmetic mean – as it leads to linear computation:

$$\Phi_k(X_k) = \frac{1}{|X_k|} \sum_{x \in X_k} \phi(x) \tag{10}$$

Given this form, the local normalisation factor represented in the denominator of Eq. (7) can now efficiently represented as the sum of all weighted sums of objects, exploiting the symmetry. Since each object x in the remainder set R_k participates in the *same* additive manner towards the construction of the denominator in Eq. (7), it must admit the following form[10]:

$$Z_k = \sum_{S \in 2^{R_k}} \Phi_k(S) = \sum_{S \in 2^{R_k}} \frac{1}{|S|} \sum_{x \in S} \phi(x) = C_k \sum_{x \in R_k} \phi(x) \tag{11}$$

where C_k 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_k, which happens to be $\frac{2^{N_k}-1}{N_k}$). To see this, substitute Eqs. (10) and (11) into Eq. (7):

[10] 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. (7) 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.

$$\log P\left(X_k \mid X_{1:k-1}\right) = \log \frac{\Phi_k\left(X_k\right)}{\sum\limits_{S \in 2^{R_k}} \Phi_k(S)} = \log \frac{1}{C_k\left|X_k\right|} \frac{\sum_{x \in X_k} \phi(x)}{\sum_{x \in R_k} \phi(x)} \tag{12}$$
$$= \log \frac{\sum_{x \in X_k} \phi(x)}{\sum_{x \in R_k} \phi(x)} - \log C_k\left|X_k\right|$$

Since $\log C_k\left|X_k\right|$ is a constant w.r.t the parameters used to parameterise the potential 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. (6) is equivalent to maximising each local log-likelihood function given in Eq. (12) for each k. Discarding the constant term in Eq. (12), we re-write it in this simpler form:

$$\log P\left(X_k \mid X_{1:k-1}\right) = \log \sum_{x \in X_k} g_k\left(x \mid X_{1:k-1}\right) \tag{13}$$
$$\text{where } \; g_k\left(x \mid X_{1:k-1}\right) = \frac{\phi(x)}{\sum_{x \in R_k} \phi(x)}$$

Evaluation of data log-likelihood and its gradient using dynamic programming

We now show that the gradient-based learning complexity can be linear in N. To see how, let us introduce an auxiliary array $\left\{a_k = \sum_{x \in R_k} \phi\left(x\right)\right\}_{k=1}^{K_\sigma}$. We start from the last subset, where $a_{K_\sigma} = \sum_{x \in X_{K_\sigma}} \phi\left(x\right)$, and proceed backward as $a_k = a_{k+1} + \sum_{x \in X_k} \phi\left(x\right)$ for $k < K_\sigma$. Clearly $\left\{a_1, a_2, .., a_{K_\sigma}\right\}$ can be computed in N time. Thus, $g_k\left(\cdot\right)$ in Eq. (13) can also be computed linearly via the relation $g_k\left(x\right) = \phi\left(x\right)/a_k$. 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. (6), using Eq. (13), the log-likelihood function and its gradient, without explicit mention of the parameters, can be shown to be[11]

$$\mathcal{L} = \log P\left(X_1, \ldots, X_{K_\sigma}\right) \tag{14}$$
$$= \sum_{k=1}^{K_\sigma} \log \sum_{x \in X_k} g_k\left(x \mid X_{1:k-1}\right) = \sum_{k=1}^{K_\sigma} \log \sum_{x \in X_k} \frac{\phi(x)}{a_k}$$
$$\partial\mathcal{L} = \sum_k \partial \log \sum_{x \in X_k} \phi\left(x\right) - \sum_k \partial \log a_k \tag{15}$$
$$= \sum_k \frac{\sum_{x \in X_k} \partial\phi\left(x\right)}{\sum_{x \in X_k} \phi\left(x\right)} - \sum_k \frac{1}{a_k} \sum_{x \in R_k} \partial\phi\left(x\right) \tag{16}$$

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 an 2-D auxiliary array[12] $\left\{\mathbf{b}_k = \sum_{x \in R_k} \partial\phi(x)\right\}_{k=1}^{K_\sigma}$. Again, we start from the last subset, where $\mathbf{b}_{K_\sigma} =$

[11] To be more precise, for $k = 1$ we define $X_{1:0}$ to be $\emptyset$.
[12] This is 2-D because we also need to index the parameters as well as the subsets.

$\sum_{x \in X_{K_\sigma}} \partial\phi\,(x)$, 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.

3.3.3. Maximum Aggregation

We now show that it is also efficient to compute for the cases of minimum and maximum aggregation functions[13]. We present here the case for maximum aggregation (the minimum case is similar):

$$\Phi_k\,(X_k) = \max_{x \in X_k} \phi(x)$$

The local normalisation factor in Eq. (7) reads:

$$\sum_{S \in 2^{R_k}} \Phi_k(S) = \sum_{S \in 2^{R_k}} \max_{x \in S} \phi(x) = \sum_{x \in R_k} M_k(x)\phi(x) \tag{17}$$

where $M_k(x)$ is the number of times the object x is the largest member of a subset of R_k. More precisely, $M_k(x) = 2^{N_k - n}$ for the n-th largest object of the set R_k. Clearly, for the smallest object $M_k(x) = 1$ – it is the member of the subset containing only itself. The details of calculating $M_k(x)$ are left in Appendix A.2.

Evaluation of data log-likelihood and its gradient

The data log-likelihood requires computing the sequence $\left\{Z_k = \sum_{S \in 2^{R_k}} \Phi_k(S)\right\}$ for all k. Each element costs $N_k \log N_k$ steps for sorting. Totally, the cost is $\sum_{k=1}^{K_\sigma} N_k \log N_k$.

We show that the gradient of the data log-likelihood $\partial\mathcal{L}$ can also be evaluated efficiently. Recall from in Eq. (9), we need to compute the following expectation:

$$\begin{aligned}
\sum_{S \in 2^{R_k}} P\,(S \mid X_{1:k-1})\,\partial \log \Phi_k\,(S) &= \frac{1}{Z_k} \sum_{S \in 2^{R_k}} \left[\max_{x \in S} \phi(x)\right] \partial \left[\max_{x \in S} \log \phi(x)\right] \\
&= \frac{1}{Z_k} \sum_{x \in R_k} M_k(x)\phi(x)\partial\phi(x) \\
&= \sum_{x \in R_k} g_k(x)\partial\phi(x)
\end{aligned}$$

where

$$g_k(x) = \frac{M_k(x)\phi(x)}{\sum_{x \in R_k} M_k(x)\phi(x)}$$

The time complexity for the gradient $\partial\mathcal{L}$ is then $F \times \sum_{k=1}^{K_\sigma} N_k \log N_k$.

3.3.4. Summary

Table 1 summarises the complexity of the PMOP in comparison with pairwise models (see also Appendix A.3 for details).

[13] We especially thank the reviewer who pointed out that the computation could be efficient for 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.3 for the details.

3.4. General Case of PMOP: MCMC-based Learning

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 expectation (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 in our setting: Gibbs sampling and Metropolis-Hastings sampling.

3.4.1. Inference and Learning Tasks

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

Normalising constant. To estimate the normalisation constant Z_k, we employ an efficient procedure called Annealed Importance Sampling (AIS) proposed recently [40]. 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 computed as $Z_k(1) \approx Z_k(0)\left(\sum_{r=1}^R \omega^{(r)}/R\right)$.

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 stochastic. Fortunately, it has been shown that stochastic gradient ascent can still

maximise the data likelihood if the learning rate is sufficiently small [22, 54]. There are two general ways to combine the sampling procedure and parameter update. One method, called *Contrastive Divergence* (CD) [22], maintains multiple short Markov chains, one per data instance. At each updating, the chains are restarted from the data samples and run for Δ steps then the sample is collected. This is often referred to as CD-Δ in the recent machine learning literature, where Δ can be as small as 1. The idea is that the Markov chains need only to depart from the data samples a little towards the true model distribution, and that is enough for moving the model parameters along the gradient.

The other method, also known as *Persistent Contrastive Divergence* (PCD), maintains a fixed number of Markov chains in parallel [54, 47]. Unlike the PCD, the Markov chains are not restarted but persistent for the entire sampling/updating cycles. The assumption is that The parameter update is likely to change the model quite little and the sampler can still run as if the model is still the same. However, unlike the standard application of PCD, here the models are query-specific, and thus we need to address each query separately, that is, one set of chain per query/stage. For this reason we follow the CD-1 since it is easier to implement and we do not have to maintain the chains in memory.

3.4.2. Gibbs Sampling

We first note that the problem of sampling sets can be viewed as exploring a *fully-connected Markov random field with binary variables* [19]. At stage k, each remaining object in $\{X \setminus X_{1:k-1}\}$ is attached with binary variable whose states[14] are either `selected` or `notselected` at k-th stage. 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 $(k-1)$-th stage. The set function $\Phi_k(X_k)$ is now the potential function of all selected variables. The operation is fairly simple - we 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 potential of the currently selected subset X_k if x is included and $\Phi_k(X_k^{-x})$ is when x is not. Since samples collected in this way are highly correlated, we need only to retain a sample after every fixed number of passes though all remaining objects. Other samples are discarded.

The main advantage of this method is its simplicity with no tuning parameters, but it may be slow to achieve independent samples. The pseudo code for the Gibbs routine performed at k-th stage is summarised in Algorithm 1.

3.4.3. Metropolis-Hastings Sampling

The idea is to consider a subset $S \in 2^{R_k}$ itself as a sampling state [21], and randomly move from one sample state to another. The move from S to S' is accepted with the following probability

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

[14] Please note that these states are defined for the Markov random field under study only.

Algorithm 1 Gibbs sampling for PMOP in general case.

Input: input parameters, sample collection schedule
* Randomly choose an initial subset X_k.
Repeat
* 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 potential of the currently selected subset X_k if x is included and $\Phi_k(X_k^{-x})$ is when x is not.
* Collect samples according to schedule.
Until stopping criteria met
Output: output set samples

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

Input: input parameters, sample collection schedule
* Randomly choose an initial subset X_k.
Repeat
* Randomly choose number of objects m, subject to $1 \leq m \leq N_k$.
* 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.
* Set $X_k \leftarrow S$ with the probability of

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

Collect samples according to schedule.
Until stopping criteria met
Output: output set samples

where $Q(S'|S)$ is the proposal transition probability which depends on specific proposal design, and we have cancelled the normalising constant Z_k using Eq (7). 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 need only to retain a set sample after a number of steps and discard others. The pseudo code for the Metropolis-Hastings routine is summarised in Algorithm 2.

3.4.4. Complexity Analysis

In learning using the Contrastive Divergence (CD) (described at the end of Section 3.3.1), the Markov chain needs not be converged, even though its convergence is guaranteed in the long-run [7, 22]. Rather, we need only to relax the samples away from the observed values. The CD is a proven technique and it is widely used in Markov random fields and restricted Boltzmann machines, which are a building block for deep architec-

tures [23, 38]. The time complexity is linear with a constant factor depending on the sampling (Gibbs or Metropolis-Hastings).

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 Equation 6 as

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

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 scaling parameters. 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 specific model specifications of our proposed probabilistic ordered partition models in two applications, namely Web document ranking and collaborative filtering.

4.1. Web Document Ranking

The ultimate goal of learning-to-rank is that, for each query the learnt system needs to return a list of related objects and their *ranking*[15]. Suppose 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[16], and for each query-object pair (q, x), a feature vector $x^q \in \mathbb{R}^F$ is extracted. The task is then to rank these objects in decreasing order of relevance w.r.t q. Enumerating over all possible rankings take an order of $N_q!$ time. Instead we would like to establish a *scoring 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 order of $N_q \log N_q$ instead of $N_q!$. The function specification can be a simple linear combination of features or more complicated form, such as a multilayer neural network.

[15] 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 ranking phase the system still needs to return a ranking over the list of related objects for an unseen query.

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

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 before to specify conditional model specific to q over the set of returned objects X^q as follows.

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

We can see that Eq (19) has exactly the same form of Eq (6) 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 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[17]. The value of M is typically much smaller than N_q, thus, the issue of *ties,* described at the beginning of this section, 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 scoring function $f\left(x, w\right)$ we specify the individual potential function $\phi\left(\cdot\right)$ in the exponential form: $\phi\left(x, w\right) = \exp\left\{f\left(x, w\right)\right\}$. For simplicity, assume further that the scoring function has the linear form $f(x^q, w) = w^\top x^q$. In what follows we choose two specific forms of $\Phi_k\left(X_k^q\right)$: as a geometric mean or as a arithmetic mean of individual potentials (Subsection 3.3). The geometric mean setting illustrates a particular case where we need to employ MCMC-techniques while the arithmetic mean setting leads to efficient computation.

For learning we need to compute the gradient of the log-likelihood function:

$$\partial \log P\left(X_k^q \mid X_{1:k-1}^q\right) = \partial \log \Phi_k\left(X_k^q\right) - \sum_{S_k \in 2^{R_k^q}} P\left(S_k \mid X_{1:k-1}^q\right) \partial \log \Phi_k\left(S_k\right) \quad (20)$$

Clearly the gradient depends on the specification of the set function $\Phi_k\left(X_k^q\right)$. In what follows, we detail five special cases mentioned earlier in Section 3.3.

4.1.1. Exact Cases: Maximum and Arithmetic Mean Set Functions

Maximum Set Functions

Recall from Subsection 3.3 that the set function is the max of local potentials:

$$\Phi_k\left(X_k^q\right) = \max_{x \in X_k^q} \left\{\exp\left(f(x, w)\right)\right\} \quad (21)$$

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

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

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

Arithmetic Mean Set Functions

The set function is simply the mean of local potentials:

$$\Phi_k\left(X_k^q\right) = \frac{1}{|X_k^q|}\sum_{x\in X_k^q}\exp\left\{f(x,w)\right\} \tag{22}$$

The gradient of the log-likelihood function has the following form:

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

4.1.2. Approximate Cases: Geometric Mean and Harmonic Mean Set Functions

We now consider two cases of the set functions $\Phi_k\left(X_k^q\right)$: *geometric mean* and *harmonic mean*.

Geometric mean:

Recall from Subsection 3.3 that the geometric mean set function has the following form

$$\Phi_k\left(X_k^q\right) = \left(\prod_{x\in X_k^q}\phi\left(x\right)\right)^{\frac{1}{|X_k^q|}} = \exp\left\{\frac{1}{|X_k^q|}\sum_{x\in X_k^q}f\left(x,w\right)\right\} = \exp\left\{w^\top \bar{x}_k^q\right\} \tag{23}$$

where $\bar{x}_k^q = \frac{1}{|X_k^q|}\sum_{x\in X_k} x^q$ is simply the *geometric mean* feature vector over the set X_k. This leads to a simple gradient of the log set function:

$$\frac{\partial \log \Phi_k\left(X_k^q\right)}{\partial w} = \bar{x}_k^q$$

Harmonic mean:

Recall from Subsection 3.3 that the harmonic mean set function has the following form

$$\Phi_k\left(X_k^q\right) = |X_k^q|\left[\sum_{x\in X_k^q}\frac{1}{\phi(x)}\right]^{-1} = \exp\left\{\log|X_k^q| - \log\left[\sum_{x\in X_k^q}\exp\left(-f\left(x,w\right)\right)\right]\right\}$$

Hence its gradient reads

$$\begin{aligned}\frac{\partial \log \Phi_k\left(X_k^q\right)}{\partial w} &= \left[\sum_{x\in X_k^q}\exp\left(-f\left(x,w\right)\right)\right]^{-1}\sum_{x\in X_k^q}\exp\left(-f\left(x,w\right)\right)x \\ &= \bar{x}_k^q\end{aligned}$$

where

$$\bar{x}_k^q = \sum_{x \in X_k^q} Q_k(x)x; \qquad Q_k(x) = \frac{\exp\left(-f\left(x, w\right)\right)}{\sum_{x' \in X_k^q} \exp\left(-f\left(x', w\right)\right)}$$

Updating rule:

For both the geometric and harmonic means, we need to compute the the expectation $\sum_{S_k} P(S_k \mid X_{1:k-1}^q)\bar{s}_k$ in Eq. (20). This expectation is expensive to evaluate, since there are $2^{N_k} - 1$ possible subsets. Thus, we resort to MCMC techniques (see Section 3.4) and the parameter is 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 number of samples. Typically we choose n to be small, e.g. $n = 1, 2, 3$.

4.1.3. Second-Order Features

In the case of document ranking, features are often precomputed based on prior knowledge about

- *Similarity* measures between a query and a related document, e.g. cosine between the word vectors with *tf.idf* weighting scheme, and
- *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*: We perform a Cartesian product to generate all second-order features, and
2. *Feature selection*: We first compute 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 an user u and an item i, the item worth can be chosen as follows

$$\phi_k\left(x=i,u\right)=\exp\{f(u,i)\},\ \text{ where } f(u,i)=\sum_{d=1}^{D}W_{ud}H_{di} \tag{24}$$

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 the 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 set functions, the gradient of the log-likelihood reads:

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

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

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

As the parameters are query-dependent (e.g. the number of rows in W grows with 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 worth 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

$$\hat{\mathcal{L}}=\sum_u\left\{\beta\mathcal{L}^u(W,H)+\frac{1}{2}(1-\beta)\sum_{i\in X^u}\left(r_{ui}-f\left(u,i\right)\right)^2\right\}+\lambda_1\left\|W\right\|_2^2+\lambda_2\left\|H\right\|_2^2 \tag{25}$$

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 $\left\|\cdot\right\|_2$ 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 score for every user-item pair using $f(u,i)$ in Eq (24), which then can 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 several methods resulted from our framework (see description in Section 4.1):

- *Exact learning*: PMOP-ArithM and PMOP-MAX, with arithmetic mean and maximum set functions, respectively.
- *Approximate learning*: PMOP-GeoM.Gibbs and PMOP-GeoM.MH, with geometric mean set functions under Gibbs sampling and Metropolis-Hastings sampling, respectively. Similarly, PMOP-HarmM.Gibbs is an approximate model with harmonic mean set function.

Two performance metrics are reported: the Normalised Discounted Cumulative Gain at position T (NDCG@T) [26], and the Expected Reciprocal Rank (ERR) [9]. NDCG@T metric is defined as

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

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 rank is correct. The ERR is defined as

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

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. Document Ranking

Through Web search engines, we are now able to harvest collective information about user's information need and behaviours. Through clickthrough or manually labelled data, we have large collections of annotated data, where each query is associated with a list of documents, each of which has a score indicating its relevance. Based on the query and the browsing contexts, we can extract from a query-document pair a rich set of features, capturing the information need and search behaviour. As the goal of this paper is to learn a preference model, we assume that the feature sets are given to the model.

5.1.1. Data and Settings

We employ the dataset from the recently held Yahoo! learning-to-rank challenge[18] [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

[18] This is much larger than the commonly used LETOR 3.0 and 4.0 datasets. In the preparation of this manuscript, we learnt that Microsoft had released two large sets of comparable size with that of Yahoo! but due to time constraint, we do not report the results here.

per query, the occurrences of ties are frequent. As we are mainly concerned about modelling the preferences, we use the features for each document-query pairs supplied by Yahoo!, and there are 519 unique features.

Before running algorithms, we first normalised the features across the whole training set to have mean 0 and standard deviation 1. 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). For comparison, we implement several well-known methods, including RankNet [5], linear Ranking SVM [27] and ListMLE [53]. 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[19]. Similarly, choosing quadratic loss gives us a rank regression method, which we will call Rank Regress (see Appendix A.3 for more details). From rank modelling point of view, the RankNet is essentially the Bradley-Terry model [3] applied to learning to rank. Likewise, the ListMLE is essentially the Plackett-Luce model, which has been argued to be one of the best performing methods [34].

We also implement two variants of the Bradley-Terry model with ties handling, one by Rao-Kupper [42] (denoted by PairTies-RK; this also appears to be implemented in [55] under the functional gradient setting) and another by Davidson [13] (denoted by PairTies-D; and this is the first time the Davidson method is applied to learning to rank). See Appendix A.4 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 approximate PMOPs, 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. For approximate PMOPs, we run the MCMC for a few steps per query, then update the parameter using the Stochastic Gradient Ascent. The learning is stopped after $1,000$ iterations.

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 training data. First-orders 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 (without ties handling), and our **PMOP** variants improve over ListMLE, despite of the simplicity in the potential function choices in Eqs (23) and (22). The PMOP-GeoM.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[20]. We note that the difference in the top 20 in the leaderboard[21] of the Yahoo! challenge was just 1.56%. Overall, second-

[19] 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.

[20] We are aware that the results should be associated with the error bars, however, since the data is huge, running the experiments repeatedly is extremely time-consuming.

[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.

	First-order features			Second-order features		
	ERR	NG@1	NG@5	ERR	NG@1	NG@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-ArithM	0.5038	0.7137	0.6762	0.5086	**0.7272**	**0.6858**
PMOP-ArithM(*)	**0.5054**	0.7115	0.6763	**0.5099**	0.7243	0.6835
PMOP-MAX	0.5034	0.7118	0.6707	0.5037	0.7143	0.6704
PMOP-HarmM.Gibbs	0.5018	0.7029	0.6663	0.5013	0.7026	0.6749
PMOP-GeoM.Gibbs	0.5037	0.7105	**0.6792**	0.5040	0.7124	0.6706
PMOP-GeoM.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 method and Rao-Kupper method for ties handling, respectively. PMOP-ArithM is the PMOP with arithmetic mean set functions; MAX = maximum; GeoM = Geometric mean; HarmM = Harmonic mean; MH = Metropolis-Hastings; (*) indicates that likelihood weighting in Equation 18 has been used. See Section 4.1 for detailed description.

order features help to improve performance of exact models. The PMOP-ArithM and PMOP-MAX are highly competitive against approximate methods.

This is highly significant because their time complexity is linear and the convergence is guaranteed. The numerical training time also confirmed that the PMOP-ArithM and PMOP-MAX are the fastest. All other pairwise methods are quadratic in query size, and thus numerically slower (Table 1). Complexity for approximate cases are also linear in the query size, by a constant factor that is determined by the number of iterations.

5.2. Collaborative Filtering for Movie Recommendation

In this experiment, we study the applicability of our proposed models on the task of recommending movies to user.

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 nearest integers. In this setting, the user plays the role of the query, and the movies the role of documents. 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 general audience generally agree about their ratings. To make the data more challenging, we first remove about half of 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. The low entropy means that the

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

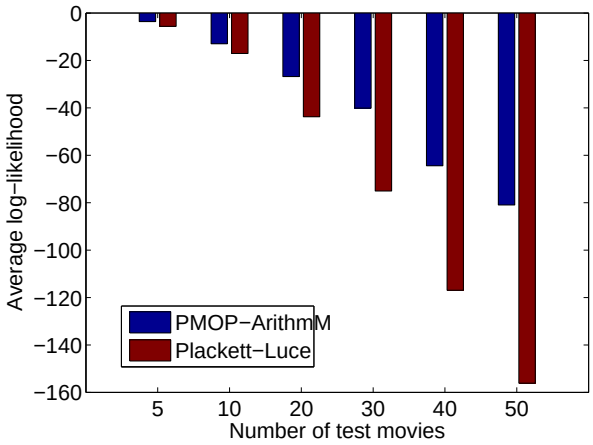

Figure 2. Log-likelihood on test movies. The ordering for Plackett-Luce is based on a sort algorithm, where ties may be broken arbitrarily. The higher likelihood, the better fitting.

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 [52] on the data with the code provided by the authors[23]. In Eq (25), 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-ArithM 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-ArithM (with tie handling) compared with the standard Plackett-Luce (without tie handling) in terms of data likelihood. For remaining movies for each user, we randomly picked $M \in \{5, 10, 20, 30, 40, 50\}$ movies for model fitting evaluation (those who number of remaining movies is less than M is not included). Figure 2 shows that PMOP-ArithM fits the 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 large number of movies.

We then compare the predictive performance of the PMOP-ArithM for recommendation of unseen movies to users. Table 3 reports results against three variants of the Cofi^{RANK}, which is perhaps the best-known algorithm in this class of problems. It

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

	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-ArithM	**0.684**	**0.576**	**0.602**

Table 3. Results for movie recommendation.

can be seen that our PMOP-ArithM is competitive as it is as good as or better than Cofi^{RANK}.

6. Discussion

In our specific choice of the local distribution in Eq (7), we share the same idea with that of Plackett-Luce, in which the probability of choosing the subset is proportional to the subset's worth, which is realised by the subset potential. 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 set functions has an interesting interpretation. From Eq (12) 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 (6) and the choice of local distribution in Eq (7) 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

$$P\left(X_1, \ldots, X_{K_\sigma}\right) = P\left(X_{K_\sigma}\right) \prod_{k=1}^{K_\sigma - 1} P\left(X_k \mid X_{k+1:K_\sigma}\right) \tag{26}$$

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 [16] 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 (7), 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 [30]). 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 as a directed model. The undirected setting is also of great interest, but it is beyond the scope of this paper, leaving out an open research topic.

7. Conclusions

In this paper, we have addressed the problem of modelling human preferences with ties. Our first contribution is the casting of this problem into a more generic probabilistic formulation of set partitioning and ordering. This resulted in a hyper-exponential state-space which grows as quick 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 are chosen at each stage, thus reducing the state-space significantly to $2^{N_k} - 1$ 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 thousand objects. We also proved that there exists highly interpretable cases where learning can be carried out in *linear* time. We evaluated the proposed models on two problems: the first is document ranking with the large-scale data from the recently held 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 are rooms for future advancement of the current work. First, 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. There are 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.

Acknowledgements

We thank anonymous reviewers for constructive comments, in particular, the suggestion of minimum/maximum aggregation functions.

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

A. Appendix

A.1. Computing C_k

We here calculate the constant C_k in Eq (11). Let us rewrite the equation for ease of comprehension

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

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

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

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

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

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 the imagine a collection of N_k variables, each has two states: `selected` and `notselected`, where `selected` means 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 the 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 (11)), the the contribution towards C_k is then $\binom{N_k}{m}\frac{1}{N_k}$. Finally, we can compute the constant C_k, which is the weighted number of times an object belongs to any subset of any size, as follows

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

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

A.2. Computing $M_k(x)$

We now calculate the constant $M_k(x)$ in Eq. (17), which is reproduced here for clarity:

$$\sum_{S\in 2^{R_k}} \max_{x\in S} \phi(x) = \sum_{x\in R_k} M_k(x)\phi(x) \tag{27}$$

First we arrange the objects in the *decreasing* order of worth $\phi(x)$. For notation convenience we assume that the order is $1, 2, 3, ..., N_k$. The largest object will appear in a subset consisting of only itself, and $2^{N_k-1}-1$ other subsets. Thus $M_k(1) = 2^{N_k-1}$ since for all subsets to which the largest object belong, the maximum aggregation is the worth of the object, as per definition. Now removing the largest object, consider the second largest one. With the same argument as before, $M_k(2) = 2^{N_k-2}$. Continuing the same line of reasoning, we end up $M_k(n) = 2^{N_k-n}$.

A.3. 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 relevancy of the 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:

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

However, these losses behave quite different from each other. For the RankNet and Rank Boost, 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 the first sight, 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.4. Learning the Pairwise Ties Models

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

A.4.1. Davidson method

Recall from Section 2 that in the Davidson 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 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

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

A.4.2. Rao-Kupper method

Recall from Section 2 that the Rao-Kupper 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

$$\frac{\partial \log P(x_i \succ x_j; w)}{\partial w} = (1 - P_i)\frac{\partial \log \phi(x_i, w)}{\partial w} - (1 + e^{\alpha})P_j \frac{\partial \log \phi(x_j, w)}{\partial w}$$

$$\frac{\partial \log P(x_i \succ x_j; w)}{\partial \alpha} = -P_j e^{\alpha}$$

$$\frac{\partial \log P(x_i \sim x_j; w)}{\partial w} = (1 - P_i - (1 + e^{\alpha})P_i^*)\frac{\partial \log \phi(x_i, w)}{\partial w} + (1 - P_j - (1 + e^{\alpha})P_j^*)\frac{\partial \log \phi(x_j, w)}{\partial w}$$

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

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] L. Becchetti, U.M. Colesanti, A. Marchetti-Spaccamela, and A. Vitaletti. Recommending items in pervasive scenarios: models and experimental analysis. *Knowledge and Information Systems*, pages 1–24, 2010.

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

[4] 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.

[5] 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.

[6] 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.

[7] M. A. Carreira-Perpiñán and Geoffrey E. Hinton. On contrastive divergence learning. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS)*, pages 33–40, Barbados, Jan 6-8 2005. Society for Artificial Intelligence and Statistics.

[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] W. Chu and Z. Ghahramani. Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6(1):1019, 2006.

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

[12] D. Cossock and T. Zhang. Statistical analysis of Bayes optimal subset ranking. *IEEE Transactions on Information Theory*, 54(11):5140–5154, 2008.

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

[17] 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.

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

[19] 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.

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

[21]W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
[22]G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.
[23]G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
[24]J. Huang, C. Guestrin, and L. Guibas. Fourier theoretic probabilistic inference over permutations. *The Journal of Machine Learning Research*, 10:997–1070, 2009.
[25]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.
[26]K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):446, 2002.
[27]T. Joachims. Optimizing search engines using clickthrough data. In *Proc. of SIGKDD*, pages 133–142. ACM New York, NY, USA, 2002.
[28]M.G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.
[29]Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, 2008.
[30]S.L. Lauritzen. *Graphical Models*. Oxford Science Publications, 1996.
[31]G. Lebanon and Y. Mao. Non-parametric modeling of partially ranked data. *Journal of Machine Learning Research*, 9:2401–2429, 2008.
[32]C.W. Leung, S.C. Chan, and F. Chung. A collaborative filtering framework based on fuzzy association rules and multiple-level similarity. *Knowledge and Information Systems*, 10(3):357–381, 2006.
[33]N.N. Liu, M. Zhao, and Q. Yang. Probabilistic latent preference analysis for collaborative filtering. In *CIKM*, pages 759–766. ACM, 2009.
[34]T.Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.
[35]R.D. Luce. *Individual choice behavior*. Wiley New York, 1959.
[36]C.L. Mallows. Non-null ranking models. I. *Biometrika*, 44(1):114–130, 1957.
[37]J.I. Marden. *Analyzing and modeling rank data*. Chapman & Hall/CRC, 1995.
[38]Benjamin Marlin, Kevin Swersky, Bo Chen, and Nando de Freitas. Inductive Principles for Restricted Boltzmann Machine Learning. In *Proceedings of the 13rd International Conference on Artificial Intelligence and Statistics*, Chia Laguna Resort, Sardinia, Italy, May 2010.
[39]M. Mureşan. *A concrete approach to classical analysis*. Springer Verlag, 2008.
[40]R.M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.
[41]R.L. Plackett. The analysis of permutations. *Applied Statistics*, pages 193–202, 1975.
[42]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.
[43]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.
[44]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.
[45]Y. Shi, M. Larson, and A. Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *ACM RecSys*, pages 269–272. ACM, 2010.
[46]C. Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.
[47]T. Tieleman and G. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM New York, NY, USA, 2009.
[48]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.
[49]J.H. van Lint and R.M. Wilson. *A course in combinatorics*. Cambridge Univ Pr, 1992.
[50]S. Vembu and T. Gärtner. Label ranking algorithms: A survey. *Preference Learning*, page 45, 2010.
[51]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.
[52]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.
[53]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.

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

[55]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.