

# Low-Rank Factorization of Determinantal Point Processes for Recommendation

Mike Gartrell  
Microsoft  
mike.gartrell@acm.org

Ulrich Paquet<sup>\*</sup>  
Microsoft  
ulripa@microsoft.com

Noam Koenigstein  
Microsoft  
noamko@microsoft.com

## ABSTRACT

Determinantal point processes (DPPs) have garnered attention as an elegant probabilistic model of set diversity. They are useful for a number of subset selection tasks, including product recommendation. DPPs are parametrized by a positive semi-definite kernel matrix. In this work we present a new method for learning the DPP kernel from observed data using a low-rank factorization of this kernel. We show that this low-rank factorization enables a learning algorithm that is nearly an order of magnitude faster than previous approaches, while also providing for a method for computing product recommendation predictions that is far faster (up to 20x faster or more for large item catalogs) than previous techniques that involve a full-rank DPP kernel. Furthermore, we show that our method provides equivalent or sometimes better predictive performance than prior full-rank DPP approaches, and better performance than several other competing recommendation methods in many cases. We conduct an extensive experimental evaluation using several real-world datasets in the domain of product recommendation to demonstrate the utility of our method, along with its limitations.

## 1. INTRODUCTION

Subset selection problems arise in a number of applications, including recommendation [9], document summarization [16, 21], and Web search [15]. In these domains, we are concerned with selecting a good subset of high-quality items that are distinct. For example, a recommended subset of products presented to a user should have high predicted ratings for that user while also being diverse, so that we increase the chance of capturing the user’s interest with at least one of the recommended products.

Determinantal point processes (DPPs) offer an attractive model for such tasks, since they jointly model *set diversity* and *quality or popularity*, while offering a compact parameterization and efficient algorithms for performing inference. A distribution over sets that encourages diversity is of particular interest when recommendations are complementary; for example, when a shopping basket contains a laptop and a carrier bag, a complementary addition to the basket would typically be a laptop cover, rather than another laptop.

DPPs can be parameterized by a  $M \times M$  positive semi-definite  $\mathbf{L}$  matrix, where  $M$  is the size of the item catalog. There has been some work focused on learning DPPs from observed data consisting of example subsets [1, 9, 16, 24],

which is a challenging learning task that is conjectured to be NP-hard [17]. The most recent of this work has involved learning a nonparametric full-rank  $\mathbf{L}$  matrix [9, 24] that does not constrain  $\mathbf{L}$  to take a particular parametric form, which becomes problematic with large item catalogs, as we will see in this paper. In contrast, we present a method for learning a low-rank factorization of  $\mathbf{L}$ , which scales much better than full-rank approaches and in some cases provides better predictive performance. The scalability improvements allow us to train our model on larger datasets that are infeasible with a full-rank DPP, while also opening the door to computing online recommendations as required for many real-world applications.

In addition to the applications mentioned above, DPPs have been used for a variety of machine learning tasks [12, 14, 17, 18, 29]. We focus on the recommendation task of “basket completion” in this work, where we compute predictions for the next item that should be added to a shopping basket, given a set of items already present in the basket. This task is at the heart of online retail experiences, such as the Microsoft Store.<sup>1</sup>

Our work makes the following contributions:

1. We present a low-rank DPP model, including algorithms for learning from observed data and computing predictions in the basket-completion scenario.
2. We perform a detailed experimental evaluation of our model on several real-world datasets, and show that our approach scales substantially better than a full-rank DPP model, while providing equivalent or better predictive performance than the full-rank model. We attribute our improvements in predictive performance to the novel use of regularization in our model.
3. In addition to comparing our approach to a full-rank DPP, we also compare to several other models for basket completion and show significant improvements in predictive performance in many cases.

Section 2 gives a formal description of our model and describes our algorithms for learning and prediction. In Section 3 we present a detailed evaluation of our model in terms of predictive performance, and training and prediction run time. We survey related work in Section 4.

<sup>\*</sup>Currently at Apple.

<sup>1</sup>[www.microsoftstore.com](http://www.microsoftstore.com)

## 2. MODEL

### 2.1 Background

A DPP is a distribution over configurations of points.<sup>2</sup> In this paper we deal only with discrete DPPs, which describe a distribution over a discrete ground set of items  $\mathcal{Y} = 1, 2, \dots, M$ , which we also call the item catalog. A discrete DPP on  $\mathcal{Y}$  is a probability measure  $\mathcal{P}$  on  $2^{\mathcal{Y}}$  (the power set or set of all subsets of  $\mathcal{Y}$ ), such that for any  $Y \subseteq \mathcal{Y}$ , the probability  $\mathcal{P}(Y)$  is specified by  $\mathcal{P}(Y) \propto \det(\mathbf{L}_Y)$ . In the context of basket completion,  $\mathcal{Y}$  is the item catalog (inventory of items on sale), and  $Y$  is the subset of items in a user’s basket; there are  $2^{|\mathcal{Y}|}$  possible baskets. The notation  $\mathbf{L}_Y$  denotes the principal submatrix of the DPP kernel  $\mathbf{L}$  indexed by the items in  $Y$ . Intuitively, the diagonal entry  $L_{ii}$  of the kernel matrix  $\mathbf{L}$  captures the importance or quality of item  $i$ , while the off-diagonal entry  $L_{ij} = L_{ji}$  measures the similarity between items  $i$  and  $j$ .

The normalization constant for  $\mathcal{P}$  follows from the observation that  $\sum_{Y' \subseteq \mathcal{Y}} \det(\mathbf{L}_{Y'}) = \det(\mathbf{L} + \mathbf{I})$ . The value  $\det(\mathbf{L}_Y)$  associates a “volume” to basket  $Y$ , and its probability is normalized by the “volumes” of all possible baskets  $Y' \subseteq \mathcal{Y}$ . Therefore, we have

$$\mathcal{P}(Y) = \frac{\det(\mathbf{L}_Y)}{\det(\mathbf{L} + \mathbf{I})}. \quad (1)$$

We use a low-rank factorization of the  $M \times M$   $\mathbf{L}$  matrix,

$$\mathbf{L} = \mathbf{V}\mathbf{V}^T, \quad (2)$$

for the  $M \times K$  matrix  $\mathbf{V}$ , where  $M$  is the number of items in the item catalog and  $K$  is the number of latent trait dimensions. As we shall see in this paper, this low-rank factorization of  $\mathbf{L}$  leads to significant efficiency improvements compared to a model that uses a full-rank  $\mathbf{L}$  matrix when it comes to model learning and computing predictions. This also places an implicit constraint on the space of subsets of  $\mathcal{Y}$ , since the model is restricted to place zero probability mass on subsets with more than  $K$  items (all eigenvalues of  $\mathbf{L}$  beyond  $K$  are zero). We see this from the observation that a sample from a DPP will not be larger than the rank of  $\mathbf{L}$  [8].

### 2.2 Learning

Our learning task is to fit a DPP kernel  $\mathbf{L}$  based on a collection of  $N$  observed subsets  $\mathcal{A} = \{A_1, \dots, A_n\}$  composed of items from the item catalog  $\mathcal{Y}$ . These observed subsets  $\mathcal{A}$  constitute our training data, and our task is to maximize the likelihood for data samples drawn from the same distribution as  $\mathcal{A}$ . The log-likelihood for seeing  $\mathcal{A}$  is

$$f(\mathbf{V}) = \log \mathcal{P}(\mathcal{A}|\mathbf{V}) = \sum_{n=1}^N \log \mathcal{P}(A_n|\mathbf{V}) \quad (3)$$

$$= \sum_{n=1}^N \log \det(\mathbf{L}_{[n]}) - N \log \det(\mathbf{L} + \mathbf{I}) \quad (4)$$

<sup>2</sup>DPPs originated in statistical mechanics [23], where they were used to model distributions of fermions. Fermions are particles that obey the Pauli exclusion principle, which indicates that no two fermions can occupy the same quantum state. As a result, systems of fermions exhibit a repulsion or “anti-bunching” effect, which is described by a DPP. This repulsive behavior is a key characteristic of DPPs, which makes them a capable model for diversity.

where  $[n]$  indexes the observations or objects in  $\mathcal{A}$ . We call the log-likelihood function  $f$ , to avoid confusion with the matrix  $\mathbf{L}$ . Recall from (2) that  $\mathbf{L} = \mathbf{V}\mathbf{V}^T$ .

The next two subsections describe how we perform optimization and regularization for learning the DPP kernel.

### 2.3 Optimization Algorithm

We determine the  $\mathbf{V}$  matrix by gradient ascent. Therefore, we want to quickly compute the derivative  $\partial f / \partial \mathbf{V}$ , which will be a  $M \times K$  matrix. For  $i \in 1, \dots, M$  and  $k \in 1, \dots, K$ , we need a matrix of scalar derivatives,

$$\left\{ \frac{\partial f}{\partial \mathbf{V}} \right\}_{ik} = \frac{\partial f}{\partial v_{ik}}.$$

Taking the derivative of each term of the log-likelihood, we have

$$\begin{aligned} \frac{\partial f}{\partial v_{ik}} &= \sum_{n:i \in [n]} \frac{\partial}{\partial v_{ik}} \left( \log \det(\mathbf{L}_{[n]}) \right) - N \frac{\partial}{\partial v_{ik}} \left( \log \det(\mathbf{L} + \mathbf{I}) \right) \\ &= \sum_{n:i \in [n]} \text{tr} \left( \mathbf{L}_{[n]}^{-1} \frac{\partial \mathbf{L}_{[n]}}{\partial v_{ik}} \right) - N \text{tr} \left( (\mathbf{L} + \mathbf{I})^{-1} \frac{\partial (\mathbf{L} + \mathbf{I})}{\partial v_{ik}} \right). \end{aligned} \quad (5)$$

To compute the first term of the derivative, we see that

$$\text{tr} \left( \mathbf{L}_{[n]}^{-1} \frac{\partial \mathbf{L}_{[n]}}{\partial v_{ik}} \right) = \mathbf{a}_i \cdot \mathbf{v}_k + \sum_{j=1}^M a_{ji} v_{jk}, \quad (6)$$

where  $\mathbf{a}_i$  denotes row  $i$  of the matrix  $\mathbf{A} = \mathbf{L}_{[n]}^{-1}$  and  $\mathbf{v}_k$  denotes column  $k$  of  $\mathbf{V}_{[n]}$ . Note that  $\mathbf{L}_{[n]} = \mathbf{V}_{[n]}\mathbf{V}_{[n]}^T$ . Computing  $\mathbf{A}$  is a usually a relatively inexpensive operation, since the number of items in each training instance  $A_n$  is generally small for many recommendation applications.

To compute the second term of the derivative, we see that

$$\text{tr} \left( (\mathbf{L} + \mathbf{I})^{-1} \frac{\partial (\mathbf{L} + \mathbf{I})}{\partial v_{ik}} \right) = \mathbf{b}_i \cdot \mathbf{v}_k + \sum_{j=1}^M b_{ji} v_{jk} \quad (7)$$

where  $\mathbf{b}_i$  denotes row  $i$  of the matrix  $\mathbf{B} = \mathbf{I}_m - \mathbf{V}(\mathbf{I}_k + \mathbf{V}^T\mathbf{V})^{-1}\mathbf{V}^T$ . Computing  $\mathbf{B}$  is a relatively inexpensive operation, since we are inverting a  $K \times K$  matrix with cost  $O(K^3)$ , and  $K$  (the number of latent trait dimensions) is usually set to a small value.

#### 2.3.1 Stochastic Gradient Ascent

We implement stochastic gradient ascent with a form of momentum known as Nesterov’s Accelerated Gradient (NAG) [26]:

$$\mathbf{W}_{t+1} = \beta \mathbf{W}_t + (1 - \beta) * \epsilon \nabla f(\mathbf{V}_t + \beta \mathbf{W}_t) \quad (8)$$

$$\mathbf{V}_{t+1} = \mathbf{V}_t + \mathbf{W}_{t+1} \quad (9)$$

where  $\mathbf{W}$  accumulates the gradients,  $\epsilon > 0$  is the learning rate,  $\beta \in [0, 1]$  is the momentum/NAG coefficient, and  $\nabla f(\mathbf{V} + \beta \mathbf{W}_t)$  is the gradient at  $\mathbf{V} + \beta \mathbf{W}_t$ .

We use the following schedule for annealing the learning rate:

$$\epsilon_t = \frac{\epsilon_0}{1 + t/T} \quad (10)$$

where  $\epsilon_0$  is the initial learning rate,  $t$  is the iteration counter, and  $T$  is number of iterations for which  $\epsilon$  should be kept nearly constant. This serves to keep  $\epsilon$  nearly constant for the first  $T$  training iterations, which allows the algorithm to

find the general location of the local maximum, and then anneals  $\epsilon$  at a slow rate that is known from theory to guarantee convergence to a local maximum [28]. In practice, we set  $T$  so that  $\epsilon$  is held nearly fixed until the iteration just before the test log-likelihood begins to decrease (which indicates that we have likely “jumped” past the local maximum), and we find that setting  $\beta = 0.95$  and  $\epsilon_0 = 1.0 \times 10^{-5}$  works well for the datasets used in this paper. Instead of computing the gradient using a single training instance for each iteration, we compute the gradient using more than one training instance, called a “mini-batch”. We find that a mini-batch size of 1000 instances works well for the datasets we tested.

## 2.4 Regularization

We add a quadratic regularization term to the log-likelihood, based on item popularity, to discourage large parameter values and avoid overfitting. Since not all items in the item catalog are purchased with the same frequency, we encode prior assumptions into the regularizer. The motivation for using item popularity in the regularizer is that the magnitude of the  $K$ -dimensional item vector can be interpreted as the popularity of the item, as shown in [8, 17].

$$f(\mathbf{V}) = \sum_{n=1}^N \log \det(\mathbf{L}_{[n]}) - N \log \det(\mathbf{L} + \mathbf{I}) - \frac{\alpha}{2} \sum_{i=1}^M \lambda_i \|\mathbf{v}_i\|^2 \quad (11)$$

where  $\mathbf{v}_i$  is the row vector from  $\mathbf{V}$  for item  $i$ , and  $\lambda_i$  is an element from a vector  $\boldsymbol{\lambda}$  whose elements are inversely proportional to item popularity,

$$\boldsymbol{\lambda} = \left( \frac{1}{C(1)}, \frac{1}{C(2)}, \dots, \frac{1}{C(i)} \right), \quad (12)$$

where  $C(i)$  is the number of occurrences of item  $i$  in the training data.

Taking the derivative of each term of the log-likelihood with this regularization term, we now have

$$\begin{aligned} \frac{\partial f}{\partial v_{ik}} &= \sum_{n:i \in [n]} \frac{\partial}{\partial v_{ik}} \left( \log \det(\mathbf{L}_{[n]}) \right) - N \frac{\partial}{\partial v_{ik}} \left( \log \det(\mathbf{L} + \mathbf{I}) \right) \\ &\quad - \frac{\alpha}{2} \sum_{i=1}^M \lambda_i \frac{\partial}{\partial v_{ik}} \left( \|\mathbf{v}_i\|^2 \right) \\ &= \sum_{n:i \in [n]} \text{tr} \left( \mathbf{L}_{[n]}^{-1} \frac{\partial \mathbf{L}_{[n]}}{\partial v_{ik}} \right) - N \text{tr} \left( (\mathbf{L} + \mathbf{I})^{-1} \frac{\partial (\mathbf{L} + \mathbf{I})}{\partial v_{ik}} \right) \\ &\quad - \alpha \lambda_i v_{ik}. \end{aligned} \quad (13)$$

## 2.5 Predictions

We seek to compute singleton next-item predictions, given a set of observed items. An example of this class of problem is “basket completion”, where we seek to compute predictions for the next item that should be added to shopping basket, given a set of items already present in the basket.

We use a  $k$ -DPP to compute next-item predictions. A  $k$ -DPP is a distribution over all subsets  $Y \in \mathcal{Y}$  with cardinality  $k$ , where  $\mathcal{Y}$  is the ground set, or the set of all items in the item catalog. Next item predictions are done via a conditional density. We compute the probability of the observed basket  $A$ , consisting of  $k$  items. For each possible item to be recommended, given the basket, the basket is enlarged with the new item to  $k + 1$  items. For the new item,

we determine the probability of the new set of  $k + 1$  items, given that  $k$  items are already in the basket. This machinery is also applicable when recommending a set  $B$ , which may contain more than one added item, to the basket.

A  $k$ -DPP is obtained by conditioning a standard DPP on the event that the set  $Y$ , a random set drawn according to the DPP, has cardinality  $k$ . Formally, for the  $k$ -DPP  $\mathcal{P}^k$  we have:

$$\mathcal{P}^k(Y) = \frac{\det(\mathbf{L}_Y)}{\sum_{|Y'|=k} \det(\mathbf{L}_{Y'})} \quad (14)$$

where  $|Y| = k$ . Unlike (1), the normalizer sums *only* over sets that have cardinality  $k$ .

As shown in [17], we can condition a  $k$ -DPP on the event that all of the elements in a set  $A$  are observed. We use  $\mathbf{L}^A$  to denote the kernel matrix for this conditional  $k$ -DPP (the same notation is used for the conditional kernel of the corresponding DPP, since the kernels are the same); we show in Section 2.5.1 how to efficiently compute this conditional kernel. For a set  $B$  not intersecting with  $A$ , where  $|A| + |B| = k$  we have:

$$\mathcal{P}^k(\mathbf{Y} = A \cup B | A \subseteq \mathbf{Y}) \propto \mathcal{P}_L^k(\mathbf{Y} = A \cup B) \quad (15)$$

$$\propto \mathcal{P}(\mathbf{Y} = A \cup B) \quad (16)$$

$$\propto \det(\mathbf{L}_B^A) \quad (17)$$

$$= \frac{\det(\mathbf{L}_B^A)}{Z_{k-|A|}^A} \quad (18)$$

where here  $B$  is a singleton set containing the possible next item for which we would like to compute a predictive probability.  $\mathbf{L}_B^A$  denotes the principal submatrix of  $\mathbf{L}^A$  indexed by the items in  $B$ .

Ref. [17] shows that the kernel matrix  $\mathbf{L}^A$  for a conditional DPP is

$$\mathbf{L}^A = \left( [(\mathbf{L} + \mathbf{I}_{\bar{A}})^{-1}]_{\bar{A}} \right)^{-1} - \mathbf{I} \quad (19)$$

where  $[(\mathbf{L} + \mathbf{I}_{\bar{A}})^{-1}]_{\bar{A}}$  is the restriction of  $(\mathbf{L} + \mathbf{I}_{\bar{A}})^{-1}$  to the rows and columns indexed by elements in  $\mathcal{Y} - A$ , and  $\mathbf{I}_{\bar{A}}$  is the matrix with ones in the diagonal entries indexed by elements of  $\mathcal{Y} - A$  and zeroes everywhere else.

The normalization constant for Eq. 18 is

$$Z_{k-|A|}^A = \sum_{\substack{|Y'|=k-|A| \\ A \cap Y' = \emptyset}} \det(\mathbf{L}_{Y'}^A), \quad (20)$$

where the sum runs over all sets  $Y'$  of size  $k - |A|$  that are disjoint from  $A$ . How can we compute it analytically?

We see from [17] that

$$Z_k = \sum_{|Y'|=k} \det(\mathbf{L}_{Y'}) = e_k(\lambda_1, \lambda_2, \dots, \lambda_M) \quad (21)$$

where  $\lambda_1, \lambda_2, \dots, \lambda_M$  are the eigenvalues of  $\mathbf{L}$  and  $e_k(\lambda_1, \lambda_2, \dots, \lambda_M)$  is the  $k$ th elementary symmetric polynomial on  $\lambda_1, \lambda_2, \dots, \lambda_M$ .<sup>3</sup>

Therefore, to compute the conditional probability for a single item  $b$  in singleton set  $B$ , given the appearance of

<sup>3</sup>Recall that when  $\mathbf{L} = \mathbf{V}\mathbf{V}^T$  is defined in a low-rank form, then all eigenvalues  $\lambda_i = 0$  for  $i > K$ , greatly simplifying the computation. When  $\mathbf{L}$  is full rank, this is not the case. Section 3 compares the practical performance of a full-rank and low-rank  $\mathbf{L}$ .

items in a set  $A$ , we have

$$\mathcal{P}_L^k(\mathbf{Y} = A \cup B | A \subseteq \mathbf{Y}) = \frac{\det(\mathbf{L}_B^A)}{Z_{k-|A|}^A} \quad (22)$$

$$= \frac{L_{bb}^A}{Z_1^A} \quad (23)$$

$$= \frac{L_{bb}^A}{e_1(\lambda_1^A, \lambda_2^A, \dots, \lambda_N^A)} \quad (24)$$

where  $\lambda_1^A, \lambda_2^A, \dots, \lambda_N^A$  are the eigenvalues of  $\mathbf{L}^A$  and  $e_1(\lambda_1^A, \lambda_2^A, \dots, \lambda_N^A)$  is the first elementary symmetric polynomial on these eigenvalues.

### 2.5.1 Efficient DPP Conditioning

The conditional probability used for prediction (and hence set recommendation or basket completion) uses  $\mathbf{L}^A$  in Eq. 19, which requires two inversions of large matrices. These are expensive operations, particularly for a large item catalog (large  $M$ ). In this section we describe a way to efficiently condition the DPP  $\mathbf{L}$  kernel that is enabled by our low-rank factorization of  $\mathbf{L}$ .

Ref. [8] shows that for a DPP with kernel  $\mathbf{L}$ , the conditional kernel  $\mathbf{L}^A$  with minors satisfying

$$\mathcal{P}(\mathbf{Y} = Y \cup A | A \subseteq \mathbf{Y}) = \frac{\det(\mathbf{L}_Y^A)}{\det(\mathbf{L}^A + \mathbf{I})} \quad (25)$$

on  $Y \subseteq \mathcal{Y} \setminus A$ , can be computed from  $\mathbf{L}$  by the rank- $|A|$  update

$$\mathbf{L}^A = \mathbf{L}_{\bar{A}} - \mathbf{L}_{\bar{A},A} \mathbf{L}_A^{-1} \mathbf{L}_{A,\bar{A}} \quad (26)$$

where  $\mathbf{L}_{\bar{A},A}$  consists of the  $|\bar{A}|$  rows and  $A$  columns of  $\mathbf{L}$ . Substituting  $\mathbf{V}$  into Eq. 26 gives

$$\mathbf{L}^A = \mathbf{V}_{\bar{A}} \mathbf{Z}^A \mathbf{V}_A^T \quad (27)$$

where

$$\mathbf{Z}^A = \mathbf{I} - \mathbf{V}_A^T (\mathbf{V}_A \mathbf{V}_A^T)^{-1} \mathbf{V}_A. \quad (28)$$

$\mathbf{Z}^A$  is a projection matrix, and is thus idempotent:  $\mathbf{Z}^A = (\mathbf{Z}^A)^2$ . Since  $\mathbf{Z}^A$  is also symmetric, we have  $\mathbf{Z}^A = (\mathbf{Z}^A)^T$ , and substituting  $\mathbf{Z}^A = \mathbf{Z}^A (\mathbf{Z}^A)^T$  into (27) yields

$$\mathbf{L}^A = \mathbf{V}_{\bar{A}} \mathbf{Z}^A (\mathbf{Z}^A)^T \mathbf{V}_A^T \quad (29)$$

$$= \mathbf{V}^A (\mathbf{V}^A)^T \quad (30)$$

where

$$\mathbf{V}^A = \mathbf{V}_{\bar{A}} \mathbf{Z}^A. \quad (31)$$

Conditioning the DPP using Eq. 30 requires computing the inverse of a  $|A| \times |A|$  matrix, as shown in Eq. 28, which is  $O(|A|^3)$ . This is much less expensive than the matrix inversions in Eq. 19 when  $|A| \ll M$ , which we expect for most recommendation applications. For example, in online shopping applications, the size of a shopping basket ( $|A|$ ) is generally far smaller than the size of the item catalog ( $M$ ).

## 3. EVALUATION

In this section we compare the low-rank DPP model with a full-rank DPP that uses a fixed-point optimization algorithm called Picard iteration [24] for learning. We wish to showcase the advantage of low-rank DPPs in practical scenarios such as basket completion. First, we compare test log-likelihood of low-rank and full-rank DPPs and show that

the low-rank model’s ability to generalize is comparable to that of the full-rank version. We also compare the training times and prediction times of both algorithms and show a clear advantage for the low-rank model presented in this paper. Our implementations of the low-rank and full-rank DPP models are written in Julia, and we perform all experiments on a Windows 10 system with 32 GB of RAM and an Intel Core i7-4770 CPU @ 3.4 GHz.

Comparing test log-likelihood values and training time is consistent with previous studies [9, 24]. Log-likelihood values however are not always correlated with other evaluation metrics. In the recommender systems community it is usually more accepted to use other metrics such as precision@ $k$  and mean percentile rank (MPR). In this paper we also compare DPPs (low-rank and full-rank) to other competing methods using these more “traditional” evaluation metrics.

Our experiments are based on several datasets:

1. **Amazon Baby Registries** - This public dataset consists of 111,006 registries of baby products from 15 different categories (such as “feeding”, “diapers”, “toys”, etc.), where the item catalog and registries for each category are disjoint. The public dataset was obtained by collecting baby registries from amazon.com and was used by previous DPP studies [9, 24]. In particular, [9] provides an in-depth description of this dataset. To maintain consistency with prior work, we used a random split of 70% of the data for training and 30% for testing. We use  $K = 30$  trait dimensions for the low-rank DPP models trained on this data. While the Baby Registries dataset is relatively large, previous studies analyzed each of its categories separately. We maintain this approach for the sake of consistency with prior work.

We also construct a dataset composed of the concatenation of the three most popular categories: apparel, diaper, and feeding. This three-category dataset allows us to simulate data that could be observed for department stores that offer a wide range of items in different product categories. Its construction is deliberate, and concatenates three disjoint subgraphs of basket-item purchase patterns. This dataset serves to highlight differences between DPPs and models based on matrix factorization (MF), as there are no items or baskets shared between the three subgraphs. Collaborative filtering-based MF models – which model each basket and item with a latent vector – will perform poorly for this dataset, as the latent vectors of baskets and items in one subgraph could be arbitrarily rotated, without affecting the likelihood or predictive error in any of the other subgraphs. MF models are invariant to global rotations of the embedded vectors. However, for the concatenated dataset, these models are also invariant to arbitrary rotations of vectors in each disjoint subgraph for the concatenated data set, as there are no shared observations between the three categories. A global ranking based on inner products could then be arbitrarily affected by the basket and item embeddings arising from each subgraph.

The low-rank approximation presented in this paper facilitates scaling-up DPPs to much larger datasets. Therefore, we conducted experiments on two additional real-world datasets, as we explain next.

2. **MS Store** - This is a proprietary dataset composed of shopping baskets purchased in Microsoft’s Web-based

Baby Registry

Category	F-Rank	L-Rank
Furniture	-7.07391	<b>-7.00022</b>
Carseats	<b>-7.20197</b>	-7.27515
Safety	-7.08845	<b>-7.01632</b>
Strollers	<b>-7.83098</b>	-7.83201
Media	<b>-12.29392</b>	-12.39054
Health	<b>-10.09915</b>	-10.36373
Toys	<b>-11.06298</b>	-11.07322
Bath	-11.89129	<b>-11.88259</b>
Apparel	<b>-13.84652</b>	-13.85295
Bedding	<b>-11.53302</b>	-11.58239
Diaper	<b>-13.13087</b>	-13.16574
Gear	-12.30525	<b>-12.17447</b>
Feeding	-14.91297	<b>-14.87305</b>
Gifts	<b>-4.94114</b>	-4.96162
Moms	-5.39681	<b>-5.34985</b>

MS Store

	F-Rank	L-Rank
All Products	<b>-15.10</b>	-15.23

**Table 1: Average test log-likelihoods values of low-rank (L-Rank) and full-rank (F-Rank) DPPs.**

store microsoftstore.com. It consists of 243,147 purchased baskets composed of 2097 different hardware and software items. We use a random split of 80% of the data for training and 20% for testing. For the low-rank DPP model trained on this data, we use  $K = 15$  trait dimensions.

- 3. Belgian Retail Supermarket** - This is a public dataset [4, 3] composed of shopping baskets purchased over three non-consecutive time periods from a Belgian retail supermarket store. There are 88,163 purchased baskets, composed of 16,470 unique items. We use a random split of 80% of the data for training and 20% for testing. We use  $K = 76$  trait dimensions for the low-rank DPP model trained on this data.

Since we are interested in the basket completion task, which requires baskets containing at least two items, we remove all baskets containing only one item from each dataset before splitting the data into training and test sets.

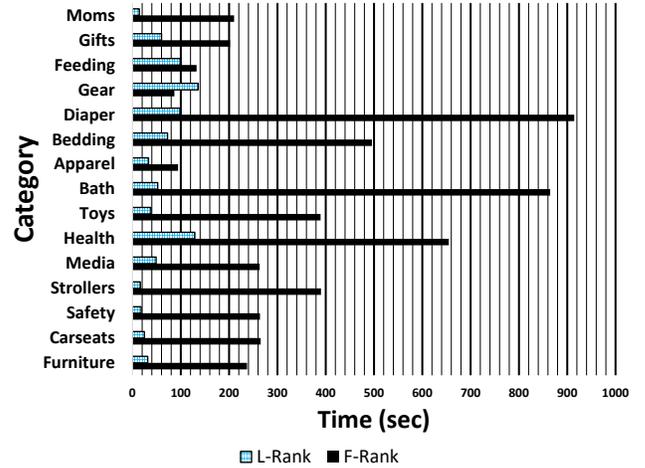
We determine convergence during training of both the low-rank and full-rank DPP models using

$$\frac{|f(\mathbf{V}_{t+1}) - f(\mathbf{V}_t)|}{|f(\mathbf{V}_t)|} \leq \delta$$

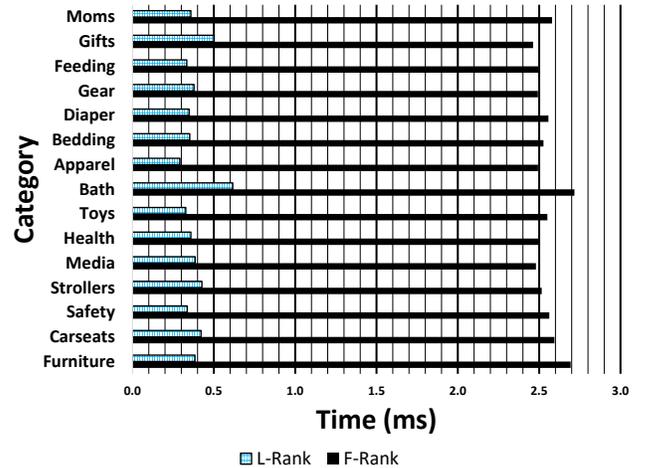
which measures the relative change in training log-likelihoods from one iteration to the next. We set  $\delta = 1.0 \times 10^{-5}$ .

### 3.1 Full Rank vs. Low Rank

We begin with comparing test log-likelihood values of the low-rank DPP model presented in this paper with the full-rank DPP trained using Picard iteration. Table 1 depicts the average test log-likelihoods values of both models across the different categories of the Baby Registries dataset as well as the MS Store dataset. In the Baby Registry dataset the full-rank model seems to perform better in 9 categories compared with 6 categories for the low-rank model, and for the MS Store dataset the full-rank model performed better. The differences in the log-likelihood values are small, and as we show in Section 3.2 these differences do not necessarily translate into better results for other evaluation metrics.



(a) Training time, in seconds



(b) Average Prediction time per basket, in milliseconds

**Figure 1: Training and prediction time of low rank DPP (L-Rank) vs. full rank DPP (F-Rank).**

#### 3.1.1 Training Time

A key contribution of the Picard iteration method was the improvement of training time (convergence time) by up to an order of magnitude [24] compared to previous methods. However, the Picard iteration method requires inverting an  $M \times M$  full-rank ( $\mathbf{L} + \mathbf{I}$ ) matrix, where  $M$  is the number of items in the catalog. This matrix inversion operation has a  $O(M^3)$  time complexity. In the low-rank model, this operation is replaced by an inversion of a  $K \times K$  matrix where  $K \ll M$ , and training is performed by stochastic gradient ascent. This translates into considerably faster training times, particularly in cases where the item catalog is large.

Figure 1(a) depicts the training time in seconds of the full-rank (F-Rank) model vs. the low-rank (L-Rank) DPP model described in this paper. Table 2 shows the number of iterations required for each model to reach convergence. Training times are shown for each of the 15 categories in the Baby Registry dataset. In all but one category, the training time of the low-rank model was considerably faster. On average, the low-rank model is 8.9 times faster to train than the full-rank model.

Category	L-Rank	F-Rank
Mom	67	1294
Gifts	126	1388
Feeding	68	123
Gear	82	136
Diaper	83	1065
Bedding	88	772
Apparel	48	129
Bath	64	1664
Toys	66	970
Health	68	1337
Media	126	958
Strollers	53	1637
Safety	59	1306
Carseats	54	1218
Furniture	54	1277

**Table 2: Number of training iterations to reach convergence, for low-rank DPP (L-Rank) and full-rank DPP (F-Rank) models**

### 3.1.2 Prediction Time

In production settings, training is usually performed in advance (offline), while predictions are computed per request (online). A typical real-world recommender system models at least thousands of items (and often much more). The “relationships” between items changes slowly with time and it is reasonable to train a model once a day or even once a week. The number of possible baskets, however, is vast and depends on the number of items in the catalog. Therefore, it is wasteful and sometimes impossible to pre-compute all possible basket recommendations in advance. The preferred choice in most cases would be to compute predictions online, in real time.

High prediction times may overload online servers, leading to high response times and even failure to provide recommendations. The ability to compute recommendations efficiently is key to any real-world recommender system. Hence, in real-world scenarios prediction times are usually much more important than training times.

Previous DPP studies [9, 24] focused on training times and did not offer any improvement in prediction times. In fact, as we show next, the average prediction time spikes for the full-rank DPP when the size of the item catalog reaches several thousand, and quickly becomes impractical in real-world settings where the inventory of items is large and fast online predictions are required. Our low-rank model facilitates far faster prediction times and scales well for large item catalogs, which is key to any practical use of DPPs. We believe this contribution opens the door to large-scale use of DPP models in commercial settings.

In Figure 1(b) we compare the average prediction time for a test-set basket for each of the 15 categories in the Baby Registry dataset. This figure shows the average time to compute predictive probabilities for all possible items that could be added to the basket for a given test basket instance, where the set of possible items are those items found in the item catalog but not in the test basket. Since the catalog is composed of a maximum of only 100 items for each Baby Registry category, due to way that the dataset was constructed, we see that these prediction times are quite small. Again we notice a clear advantage for the low-rank model across all categories: the average prediction time for the full-rank

model is 2.55 ms per basket, compared with 0.39 ms for the low-rank model (6.8 times faster). Since number of items in the catalog for each baby registry category is small (100 items), we also measured the prediction time for the MS Store dataset, which contains 2,097 items. Due to the much larger item catalog, the average time per a single basket prediction increases significantly to 1.66 seconds, which is probably too slow for many real-world recommender systems. On the other hand, the average prediction time of the low-rank model depends mostly on the number of trait dimensions in the model and takes only 83.6 ms per basket on average. These numbers indicate a speedup factor of 19.9.

Our low-rank DPP model also provides substantial savings in memory consumption as compared to the full-rank DPP. For example, the MS Store dataset, composed of a catalog of 2097 items, would require  $2097 \times 2097 \times 8$  bytes = 35.18 MB to store the full-rank DPP kernel matrix (assuming 64-bit floating point numbers), while only  $2097 \times 15 \times 8$  bytes = 251.6 KB would be required to store the low-rank  $\mathbf{V}$  matrix with  $K = 15$  trait dimensions. Therefore, the low-rank model requires approximately 140 times less memory to store the model parameters in this example, and this savings increases with larger item catalogs.

## 3.2 Basket Completion and Recommendations

Previous papers have evaluated DPP recommendations by comparing test log-likelihood values. In this section we also consider more “traditional” evaluation metrics commonly used in the recommender systems community.

We formulate the basket-completion task as follows. Let  $Y_n$  be a subset of  $n \geq 2$  co-purchased items (i.e, a basket) from the test-set. In order to evaluate the basket completion task, we pick an item  $i \in Y_n$  at random and remove it from  $Y_n$ . We denote the remaining set as  $Y_{n-1}$ . Formally,  $Y_{n-1} = Y_n \setminus \{i\}$ . Given a ground set of possible items  $\mathcal{Y} = 1, 2, \dots, M$ , we define the candidates set  $\mathcal{C}$  as the set of all items except those already in  $Y_{n-1}$ ; i.e.,  $\mathcal{C} = \mathcal{Y} \setminus Y_{n-1}$ . Our goal is to identify the missing item  $i$  from all other items in  $\mathcal{C}$ .

We compare the low-rank DPP model with the full-rank DPP model. We also consider several other competing models for the basket completion task:

- 1. Poisson Factorization (PF)** - Poisson factorization (PF) [10] is a recent variant of probabilistic matrix factorization that has been shown to work well with implicit recommendation data, such as clicks or purchases. PF models user-item interactions, such as clicks or purchases, with factorized Poisson distributions, and learns sparse, non-negative trait vectors for latent user preferences and item attributes in a low-dimensional space. Gamma priors are placed on the trait vectors; we set the gamma shape and rate hyperparameters to 0.3, following [6, 10]. The PF model is not sensitive to these settings, as indicated in [6, 10]. We use a publicly available implementation of PF [5]. (Note that [5] is actually an implementation of PF with a social component; we disable the social component for our tests, resulting in a model equivalent to PF, since our data does not involve a social graph).
- 2. Reco Matrix Factorization (RecoMF)** - RecoMF is a matrix factorization model [27] that is used as the recommendation system for Xbox Live. Sigmoid functions are used to model the odds of a user liking or disliking an

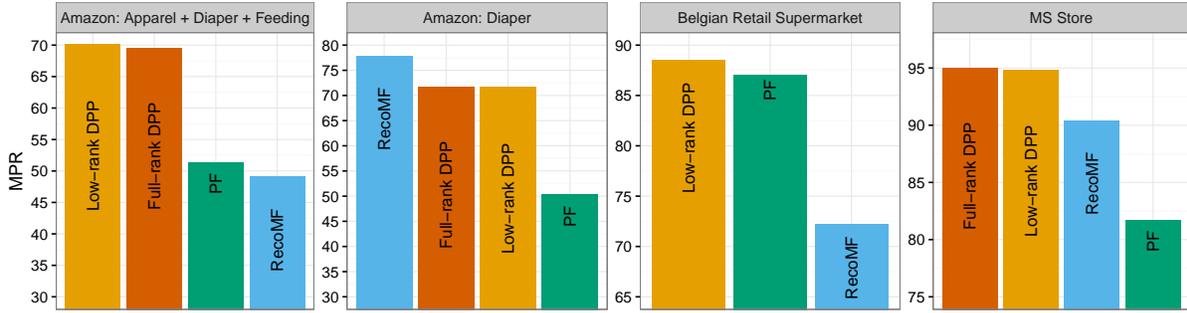


Figure 2: Mean Percentile Rank (MPR)

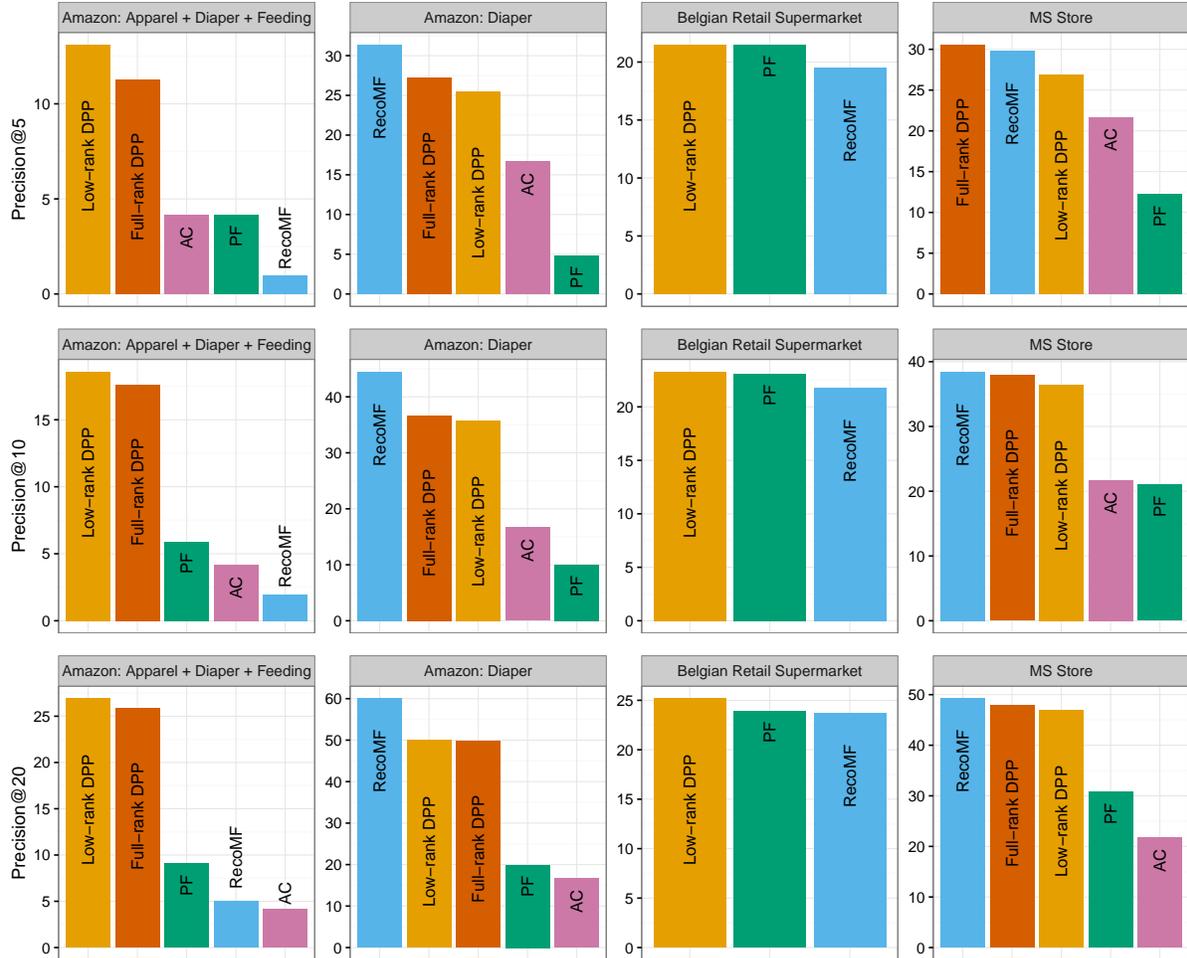


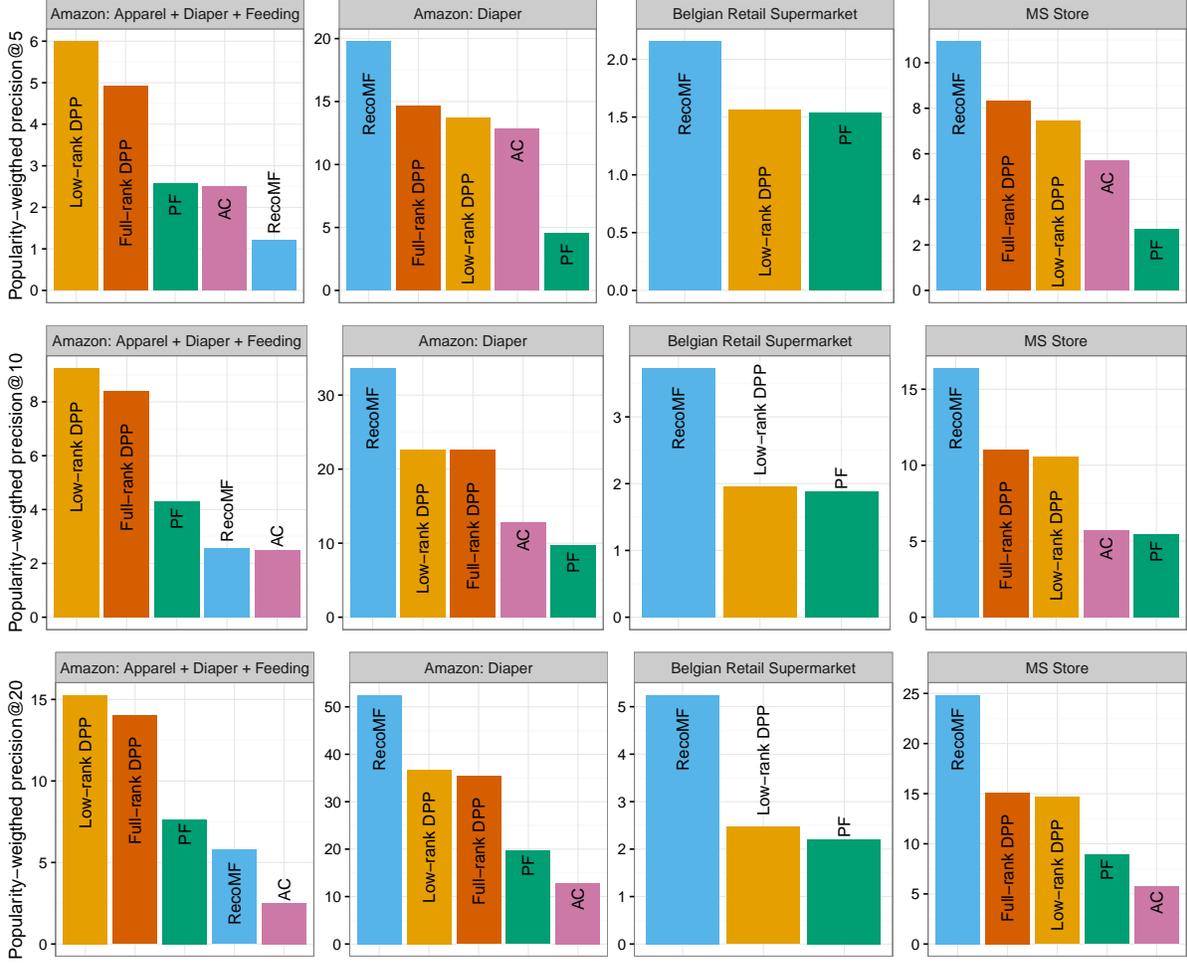
Figure 3: Precision@k

item, and RecoMF learns latent trait vectors for users and items, along with user and item biases. Unlike PF, RecoMF requires the generation of synthetic negative training instances, and uses a scheme for sampling negatives based on popularity. RecoMF places Gaussian priors on the trait vectors, and gamma hyperpriors on each. We use the hyperparameter settings described in [27], which have been found to provide good performance for implicit recommendation data.

3. **Associative Classifier (AC)** - We use an associative classifier as a competing method, since association rules

are often used for market basket analysis [2, 13]. Our associative classifier is the publicly available implementation [7] of the Classification Based on Associations (CBA) algorithm [22]. We use minimum support and minimum confidence thresholds of 1.0% and 20.0%, respectively. Since associative classifiers don't provide probability estimates for all possible sets, the model therefore cannot compute rankings for all of the candidate items in  $\mathcal{C}$ , and we therefore cannot reasonably compute MPR.

The matrix-factorization models are parameterized in terms of users and items. Since we have no explicit users in our



**Figure 4: Popularity-weighted precision@k.** These results show a limitation of the DPP models. Since this metric biases precision@k towards less popular items, we see that the RecoMF model often provides better predictive performance for less popular items.

data, we construct “virtual” users from the contents of each basket for the purposes of our evaluation, where a new user  $u_m$  is constructed for each basket  $b_m$ . Therefore, the set of items that  $u_m$  has purchased is simply the contents of  $b_m$ . Additionally, we use  $K = 40$  trait dimensions for the matrix-factorization models.

In the following evaluation we consider three measures:

1. **Mean Percentile Rank (MPR)** - Computing the Percentile Rank of an item requires the ability to rank the item  $j$  against all other items in  $\mathcal{C}$ . Therefore, the MPR evaluation results don’t include the AC model, which ranks only those items for which an association rule was found. For DPPs and other competing methods we ranked the items according to their probabilities to complete the missing set  $Y_{n-1}$ . Namely, given an item  $i$  from the candidates set  $\mathcal{C}$ , we denote by  $p_i$  the probability  $P(Y_n \cup \{i\} | Y_{n-1})$ . The Percentile Rank (PR) of the missing item  $j$  is defined by

$$\text{PR}_j = \frac{\sum_{j' \in \mathcal{C}} \mathbb{I}(p_j \geq p_{j'})}{|\mathcal{C}|} \times 100\%$$

where  $\mathbb{I}(\cdot)$  is an indicator function and  $|\mathcal{C}|$  is the number of items in the candidates set. The Mean Percentile Rank

(MPR) is the average PR of all the instances in the test-set:

$$\text{MPR} = \frac{\sum_{t \in \mathcal{T}} \text{PR}_t}{|\mathcal{T}|}$$

where  $\mathcal{T}$  is the set of test instances. MPR is a recall-oriented metric commonly used in studies that involve implicit recommendation data [11, 20]. MPR = 100 always places the held-out item for the test instance at the head of the ranked list of predictions, while MPR = 50 is equivalent to random selection.

2. **Precision@k** - We define precision@k as

$$\text{precision@k} = \frac{\sum_{t \in \mathcal{T}} \mathbb{I}[\text{rank}_t \leq k]}{|\mathcal{T}|}$$

where  $\text{rank}_t$  is the predicted rank of the held-out item for test instance  $t$ . In other words, precision@k is the fraction of instances in the test set for which the predicted rank of the held-out item falls within the top  $k$  predictions.

3. **Popularity-weighted precision@k** - Datasets used to evaluate recommendation systems typically contain a popularity bias [30], where users are more likely to provide feedback on popular items. Due to this popularity

bias, conventional metrics such as MPR and precision@ $k$  are typically biased toward popular items. Using ideas from [30], we propose popularity-weighted precision@ $k$ :

$$\text{popularity-weighted precision@}k = \frac{\sum_{t \in \mathcal{T}} w_t \mathbb{I}[\text{rank}_t \leq k]}{\sum_{t \in \mathcal{T}} w_t}$$

where  $w_t$  is the weight assigned to the held-out item for test instance  $t$ , defined as

$$w_i \propto \frac{1}{C(t)^\beta}$$

where  $C(t)$  is the number of occurrences of the held-out item for test instance  $t$  in the training data, and  $\beta \in [0, 1]$ . The weights are normalized, so that  $\sum_{j \in \mathcal{Y}} w_j = 1$ . This popularity-weighted precision@ $k$  measure assumes that item popularity follows a power-law. By assigning more weight to less popular items, for  $\beta > 0$ , this measure serves to bias precision@ $k$  towards less popular items. For  $\beta = 0$ , we obtain the conventional precision@ $k$  measure. We set  $\beta = 0.5$  in our evaluation.

Figures 2, 3, and 4 show the performance of each method and dataset for our evaluation measures. Note that we could not feasibly train the full-rank DPP or AC models on the Belgian dataset, since these models do not scale to datasets with large item catalogs. The performance of low-rank and full-rank DPP models are generally comparable on all models and metrics, with the low-rank DPP providing better performance in some cases. We attribute this advantage to the use of regularization (an informative prior, from a Bayesian perspective) in our low-rank model. We see that the RecoMF model outperforms all other models on all metrics for the Amazon Diaper dataset. For all other datasets, the low-rank DPP model outperforms on MPR by a sizeable margin, and is the only model to consistently provide high MPR across all datasets. For the precision@ $k$  metrics, the low-rank DPP often leads, or provides good performance that is close to the leader.

We see interesting results for the Amazon apparel + diaper + feeding dataset. Surprisingly, the PF and RecoMF models provide a MPR of approximately 50%, which is equivalent to basket completion by random selection. Recall that each category in the Amazon baby registry dataset is disjoint. Due to the formulation of the likelihood function for models based on matrix factorization, these models learn an embedding of item trait vectors that is mixed together across each disjoint category. This behavior results in the model mixing the predictions across each category, e.g. recommending an item from category  $A$  for a basket in category  $B$ , which is never observed in the data, thus leading to degenerate results. We empirically observe that the DPP models do not have this issue, and are able to effectively learn an embedding of items in this scenario: notice that the DPP models provide an MPR of approximately 70% for both the Amazon three-category and single-category (diaper) datasets.

### 3.2.1 Limitations

We include the popularity-weighted precision@ $k$  results in Figure 4 to highlight a limitation of the DPP models. For this metric RecoMF generally provides the best performance, with the DPP models in second place. As discussed

in [27], this behavior may result from the scheme for sampling negatives by popularity in RecoMF, which tends to improve recommendations for less popular items. We conjecture that a different regularization scheme for our low-rank DPP model, or a Bayesian version of this model that provides more robust regularization, may improve our performance on this metric. It is also important to note the limitations of this metric, including the assumption that item popularity follows a power-law, and the power-law exponent  $\beta$  setting of 0.5 used when computing the metric for each dataset. Due to these limitations, the popularity-weighted precision@ $k$  results we present here may not fully reflect the empirical popularity bias actually present in the data.

## 4. RELATED WORK

Several learning algorithms for estimating the full-rank DPP kernel matrix from observed data have been proposed. Ref. [9] presented one of the first methods for learning a non-parametric form of the DPP kernel matrix, which involves an expectation-maximization (EM) algorithm. This work also considers using projected gradient ascent on the DPP log-likelihood function, but finds that this is not a viable approach since it usually results in degenerate estimates due to the projection step.

In [24], a fixed-point optimization algorithm for DPP learning is described, called Picard iteration. Picard iteration has the advantage of being simple to implement and performing much faster than EM during training. We show in this paper that our low-rank learning approach is far faster than Picard iteration and therefore EM during training, and that our low-rank representation of the DPP kernel allows us to compute predictions much faster than any method that uses the full-rank kernel.

Ref. [1] presented Bayesian methods for learning a DPP kernel, with particular parametric forms for the similarity and quality components of the kernel. Markov chain Monte Carlo (MCMC) methods are used for sampling from the posterior distribution over kernel parameters. In contrast to this work, and similar to [9, 24], our approach uses a non-parametric form of the kernel and therefore does not assume any particular parametrization.

A method for partially learning the DPP kernel is studied in [16]. The similarity component of the DPP kernel is fixed, and a parametric form of the function for the quality component of the kernel is learned. This is a convex optimization problem, unlike the task of learning the full kernel, which is a more challenging non-convex optimization problem.

We focus on the prediction task of “basket completion” in this work, as it is at the heart of the online retail experience. For the purposes of evaluating our model, we compute predictions for the next item that should be added to a shopping basket, given a set of items already present in the basket. A number of approaches to this problem have been proposed. Ref. [25] describes a user-neighborhood-based collaborative filtering method, which uses rating data in the form of binary purchases to compute the similarity between users, and then generates a purchase prediction for a user and item by computing a weighted average of the binary ratings for that item. A technique that uses logistic regression to predict if a user will purchase an item based on binary purchase scores obtained from market basket data is described in [19]. Additionally, other collaborative filtering approaches could

be applied to the basket completion problem, such as [27], which is a one-class matrix factorization model.

## 5. CONCLUSIONS

In this paper we have presented a new method for learning the DPP kernel from observed data, which exploits the unique properties of a low-rank factorization of this kernel. Previous approaches have focused on learning a full-rank kernel, which does not scale for large item catalogs due to high memory consumption and expensive operations required during training and when computing predictions. We have shown that our low-rank DPP model is substantially faster and more memory efficient than previous approaches for both training and prediction. Furthermore, through an experimental evaluation using several real-world datasets in the domain of recommendations for shopping baskets, we have shown that our method provides equivalent or sometimes better predictive performance than prior full-rank DPP approaches, while in many cases also providing better predictive performance than competing methods.

## 6. ACKNOWLEDGEMENTS

We thank Gal Lavee and Shay Ben Elazar for many helpful discussions. We thank Nir Nice for supporting this work.

## 7. REFERENCES

- [1] R. H. Affandi, E. Fox, R. Adams, and B. Taskar. Learning the parameters of determinantal point process kernels. In *ICML*, pages 1224–1232, 2014.
- [2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of SIGMOD 1993*, pages 207–216, 1993.
- [3] T. Brijs. Retail market basket data set. In *Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, 2003.
- [4] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *KDD*, pages 254–260, 1999.
- [5] A. J. Chaney. Social Poisson factorization (SPF). <https://github.com/ajbc/spf>, 2105.
- [6] A. J. Chaney, D. M. Blei, and T. Eliassi-Rad. A probabilistic model for using social networks in personalized item recommendation. In *RecSys*, pages 43–50, 2015.
- [7] F. Coenen. TLUCS KDD implementation of CBA (classification based on associations). <http://www.csc.liv.ac.uk/~frans/KDD/Software/CMAR/cba.html>, 2004. Department of Computer Science, The University of Liverpool, UK.
- [8] J. Gillenwater. *Approximate inference for determinantal point processes*. PhD thesis, University of Pennsylvania, 2014.
- [9] J. A. Gillenwater, A. Kulesza, E. Fox, and B. Taskar. Expectation-maximization for learning determinantal point processes. In *NIPS*, pages 3149–3157, 2014.
- [10] P. Gopalan, J. M. Hofman, and D. M. Blei. Scalable recommendation with hierarchical Poisson factorization. In *UAI*, 2015.
- [11] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272, 2008.
- [12] B. Kang. Fast determinantal point process sampling with application to clustering. In *NIPS*, pages 2319–2327, 2013.
- [13] S. Kotsiantis and D. Kanellopoulos. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering*, 32(1):71–82, 2006.
- [14] A. Kulesza and B. Taskar. Structured determinantal point processes. In *NIPS*, pages 1171–1179, 2010.
- [15] A. Kulesza and B. Taskar. k-DPPs: Fixed-size determinantal point processes. In *ICML*, pages 1193–1200, 2011.
- [16] A. Kulesza and B. Taskar. Learning determinantal point processes. In *UAI*, 2011.
- [17] A. Kulesza and B. Taskar. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5(2-3):123–286, 2012.
- [18] J. T. Kwok and R. P. Adams. Priors for diversity in generative latent variable models. In *NIPS*, pages 2996–3004, 2012.
- [19] J.-S. Lee, C.-H. Jun, J. Lee, and S. Kim. Classification-based collaborative filtering using market basket data. *Expert Systems with Applications*, 29(3):700–704, 2005.
- [20] Y. Li, J. Hu, C. Zhai, and Y. Chen. Improving one-class collaborative filtering by incorporating rich user information. In *CIKM*, pages 959–968, 2010.
- [21] H. Lin and J. Bilmes. Learning mixtures of submodular shells with application to document summarization. In *UAI*, 2012.
- [22] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD*, 1998.
- [23] O. Macchi. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, pages 83–122, 1975.
- [24] Z. Mariet and S. Sra. Fixed-point algorithms for learning determinantal point processes. In *ICML*, pages 2389–2397, 2015.
- [25] A. Mild and T. Reutterer. An improved collaborative filtering approach for predicting cross-category purchases based on binary market basket data. *Journal of Retailing and Consumer Services*, 10(3):123–133, 2003.
- [26] Y. Nesterov. A method of solving a convex programming problem with convergence rate  $O(1/\sqrt{k})$ . *Soviet Mathematics Doklady*, 27:327–376, 1983.
- [27] U. Paquet and N. Koenigstein. One-class collaborative filtering with random graphs. In *WWW*, pages 999–1008, 2013.
- [28] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.
- [29] J. Snoek, R. Zemel, and R. P. Adams. A determinantal point process latent variable model for inhibition in neural spiking data. In *NIPS*, pages 1932–1940, 2013.
- [30] H. Steck. Item popularity and recommendation accuracy. In *RecSys*, pages 125–132, 2011.