Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Low-Rank Factorization of Determinantal Point Processes

Mike Gartrell[*]
Microsoft
mike.gartrell@acm.org

Ulrich Paquet[†]
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 test log-likelihood than prior full-rank DPP approaches.

1 Introduction

Subset selection problems arise in a number of applications, including recommendation (Gillenwater et al. 2014), document summarization (Kulesza and Taskar 2011b; Lin and Bilmes 2012), and Web search (Kulesza and Taskar 2011a). 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 item *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 are probabilistic submodular models 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 (Affandi et al. 2014; Gillenwater et al. 2014; Kulesza and Taskar 2011b; Mariet and Sra 2015), which is a challenging learning task that is conjectured to be NP-hard (Kulesza and Taskar 2012). Some approaches have involved learning a nonparametric full-rank $\mathbf{L}$ matrix (Gillenwater et al. 2014; Mariet and Sra 2015) that does not constrain $\mathbf{L}$ to take a particular parametric form. Full-rank DPPs scale poorly to large item catalogs, since operations such as sampling, learning, and prediction in a full-rank DPP costs approximately $O(M^3)$. This issue may be resolved for some operations (learning and/or sampling) by factorizing $\mathbf{L}$ as a Kronecker product of multiple smaller kernel matrices (Mariet and Sra 2016b), by approximate sampling techniques (Anari, Oveis Gharan, and Rezaei 2016; Li, Jegelka, and Sra 2016a; 2016b), or by defining an alternative probabilistic submodular model that scales to large M, as recently proposed in (Tschiatschek, Djolonga, and Krause 2016). In contrast, we present a method for learning a simple low-rank factorization of $\mathbf{L}$, which scales much better than full-rank DPP approaches and in some cases provides better test log-likelihood. 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 or predictions as required for many real-world applications. We also show that a low-rank factorization of $\mathbf{L}$ allows us to use a straightforward learning algorithm, such as stochastic gradient ascent, that is generally not viable for full-rank DPPs due to issues associated with projected gradient ascent required for full-rank DPP learning (Gillenwater et al. 2014).

In addition to the applications mentioned above, DPPs have been used for a variety of machine learning tasks (Affandi, Fox, and Taskar 2013; Affandi et al. 2013; Chao et al. 2015; Gillenwater, Kulesza, and Taskar 2012; Kulesza and Taskar 2010; 2012; Kwok and Adams 2012; Mariet and Sra 2016a; Snoek, Zemel, and Adams 2013). 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. In Section 2 we provide some background on DPPs, discuss how we implement learning and regularization, and describe how we efficiently compute pre-

[*]Currently at Criteo.
[†]Currently at DeepMind.

dictions. We compare the performance of our low-rank DPP model to a full-rank DPP on two real-world datasets in Section 3.

2 Model

We begin this section with some background on the formulation of DPPs and our low-rank factorization of the DPP $\mathbf{L}$ matrix, followed by a discussion of our optimization-based learning algorithm and regularization. We also describe how low-rank DPPs enable efficient computation of conditional probabilities, which is critical for quickly computing next-item predictions for the basket completion task.

2.1 Background

DPPs originated in statistical mechanics (Macchi 1975). In this paper we deal only with discrete DPPs, which describe a distribution over a discrete ground set of items $\mathcal{Y} = 1, 2, \ldots, M$, 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 $A \subseteq \mathcal{Y}$, the probability $\mathcal{P}(A)$ is specified by $\mathcal{P}(A) \propto \det(\mathbf{L}_A)$. In the context of basket completion, $\mathcal{Y}$ is the item catalog (inventory of items on sale), and A is the subset of items in a user's basket; there are $2^{|\mathcal{Y}|}$ possible baskets. The notation $\mathbf{L}_A$ denotes the principal submatrix of the DPP kernel $\mathbf{L}$ indexed by the items in A. Intuitively, the diagonal entry L_{ii} of the kernel matrix $\mathbf{L}$ captures the popularity 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 $\sum_{A' \subseteq \mathcal{Y}} \det(\mathbf{L}_{A'}) = \det(\mathbf{L} + \mathbf{I})$. Therefore, we have

$$\mathcal{P}(A) = \frac{\det(\mathbf{L}_A)}{\det(\mathbf{L} + \mathbf{I})} \ . \tag{1}$$

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

$$\mathbf{L} = \mathbf{V}\mathbf{V}^T \ , \tag{2}$$

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 $\mathbf{L}$ (Gillenwater 2014).

2.2 Learning

Our learning task is to fit a DPP kernel $\mathbf{L}$ based on a collection of N observed subsets $\mathcal{A} = \{A_1, \ldots, A_N\}$ 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

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

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

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 $i \in 1, \ldots, M$ and $k \in 1, \ldots, K$, 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

$$\frac{\partial f}{\partial v_{ik}} = \sum_{n : i \in [n]} \frac{\partial}{\partial v_{ik}} \Big(\log\det(\mathbf{L}_{[n]}) \Big) - N \frac{\partial}{\partial v_{ik}} \Big(\log\det(\mathbf{L} + \mathbf{I}) \Big)$$

$$= \sum_{n : i \in [n]} \mathsf{tr}\left(\mathbf{L}_{[n]}^{-1} \frac{\partial \mathbf{L}_{[n]}}{\partial v_{ik}} \right) - N \, \mathsf{tr}\left((\mathbf{L} + \mathbf{I})^{-1} \frac{\partial (\mathbf{L} + \mathbf{I})}{\partial v_{ik}} \right) \ . \tag{5}$$

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

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

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 $\mathbf{A}$ has cost $O(|A_n|^3)$, which results from the $|A_n| \times |A_n|$ matrix inversion and the matrix multiplication.

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

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

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 $\mathbf{B}$ has an overall cost of $O(K^3 + KM^2)$. The $O(K^3)$ cost stems from inverting a $K \times K$ matrix; the $O(KM^2)$ cost results from the matrix multiplications used to compute $\mathbf{B}$. Considering both the first and second terms of the derivative, we have a total per-iteration cost of $O(N\kappa^3 + K^3 + KM^2)$, where κ is the size of the largest observed instance in the training data. This total cost is relatively inexpensive, since κ is generally small for many recommendation applications, and K (the number of latent trait dimensions) is usually set to a relatively small value. In comparison, the per-iteration cost of the fastest known full-rank DPP learning algorithm (Mariet and Sra 2015) is $O(N\kappa^3 + M^3)$. Our low-rank DPP learning algorithm therefore scales substantially better to large item catalogs.

Stochastic Gradient Ascent We implement stochastic gradient ascent with a form of momentum known as Nesterov's Accelerated Gradient (NAG) (Nesterov 1983):

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

$$\mathbf{V}_{t+1} = \mathbf{V}_t + \mathbf{W}_{t+1} \tag{9}$$

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:

$$\epsilon_t = \frac{\epsilon_0}{1 + t/T} \tag{10}$$

where ϵ_0 is the initial learning rate, t is the iteration counter, and T is number of iterations for which ϵ should be kept nearly constant. This serves to keep ϵ nearly constant for the first T training iterations, which allows the algorithm to find the general location of the local maximum, and then anneals ϵ at a slow rate that is known from theory to guarantee convergence to a local maximum (Robbins and Monro 1951). In practice, we set T so that ϵ is held nearly fixed until the iteration just before the log-likelihood on a validation set 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. We use a minibatch size of 1000 training instances, which 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 (Gillenwater 2014; Kulesza and Taskar 2012).

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

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

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

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

$$\begin{aligned}\frac{\partial f}{\partial v_{ik}} = &\sum_{n:i\in[n]} \mathsf{tr}\left(\mathbf{L}_{[n]}^{-1}\frac{\partial\mathbf{L}_{[n]}}{\partial v_{ik}}\right) \\ &- N\,\mathsf{tr}\left((\mathbf{L}+\mathbf{I})^{-1}\frac{\partial(\mathbf{L}+\mathbf{I})}{\partial v_{ik}}\right) - \alpha\lambda_i v_{ik} .\end{aligned} \tag{13}$$

We select the regularization hyperparameter, α, using a line search performed with a validation set.

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 $\boldsymbol{Y}$, a random set drawn according to the DPP, has cardinality k. Formally, for the k-DPP $\mathcal{P}^k$ we have:

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

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

As shown in (Kulesza and Taskar 2012), 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 how to efficiently compute this conditional kernel. For a set B not intersecting with A, where $|A| + |B| = k$ we have:

$$\begin{aligned}\mathcal{P}^k(\boldsymbol{Y} = A\cup B | A \subseteq \boldsymbol{Y}) &\propto \mathcal{P}_L^k(\boldsymbol{Y} = A\cup B) &(15)\\ &\propto \mathcal{P}(\boldsymbol{Y} = A\cup B) &(16)\\ &\propto \det(\mathbf{L}_B^A) &(17)\\ &= \frac{\det(\mathbf{L}_B^A)}{Z^A_{k-|A|}} &(18)\end{aligned}$$

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. (Kulesza and Taskar 2012) shows that the kernel matrix $\mathbf{L}^A$ for a conditional DPP is

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

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

$$Z^A_{k-|A|} = \sum_{\substack{|Y'|=k-|A| \\ A\cap Y'=\emptyset}} \det(\mathbf{L}^A_{Y'}) , \tag{20}$$

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 (Kulesza and Taskar 2012) that

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

where $\lambda_1, \lambda_2, \ldots, \lambda_M$ are the eigenvalues of $\mathbf{L}$ and $e_k(\lambda_1, \lambda_2, \ldots, \lambda_M)$ is the kth elementary symmetric polynomial on $\lambda_1, \lambda_2, \ldots, \lambda_M$.[1]

Therefore, to compute the conditional probability for a single item b in singleton set B, given the appearance of items in a set A, we have

$$\mathcal{P}^k(\boldsymbol{Y} = A \cup B | A \subseteq \boldsymbol{Y}) = \frac{\det(\mathbf{L}_B^A)}{Z_{k-|A|}^A} = \frac{L_{bb}^A}{Z_1^A} \tag{22}$$

$$= \frac{L_{bb}^A}{e_1(\lambda_1^A, \lambda_2^A, \ldots, \lambda_N^A)} \tag{23}$$

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

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. (Gillenwater 2014) shows that for a DPP with kernel $\mathbf{L}$, the conditional kernel $\mathbf{L}^A$ can be computed from $\mathbf{L}$ by the rank-$|A|$ update

$$\mathbf{L}^A = \mathbf{L}_{\bar{A}} - \mathbf{L}_{\bar{A},A} \mathbf{L}_A^{-1} \mathbf{L}_{A,\bar{A}} \tag{24}$$

where $\mathbf{L}_{\bar{A},A}$ consists of the $|\bar{A}|$ rows and A columns of $\mathbf{L}$. Substituting $\mathbf{V}$ into Eq. 24 gives

$$\mathbf{L}^A = \mathbf{V}_{\bar{A}} \mathbf{Z}^A \mathbf{V}_{\bar{A}}^T \tag{25}$$

where

$$\mathbf{Z}^A = \mathbf{I} - \mathbf{V}_A^T (\mathbf{V}_A \mathbf{V}_A^T)^{-1} \mathbf{V}_A \,. \tag{26}$$

$\mathbf{Z}^A$ 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 (25) yields

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

$$= \mathbf{V}^A (\mathbf{V}^A)^T \tag{28}$$

where $\mathbf{V}^A = \mathbf{V}_{\bar{A}} \mathbf{Z}^A$.

Conditioning the DPP using Eq. 28 requires computing the inverse of a $|A| \times |A|$ matrix and several matrix multiplications, as shown in Eq. 26, which is $O(|A|^3 + K|\bar{A}|^2)$. This is much less expensive than the $O(M^3)$ 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).

[1] 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 $\mathbf{L}$.

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 (Mariet and Sra 2015) 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.

The full-rank DPP model is initialized randomly by drawing from a Wishart distribution, as described in (Mariet and Sra 2015). The low-rank DPP is also initialized randomly by drawing each entry in the $\mathbf{V}$ matrix from a uniform(0, 1) distribution, with the same random seed used to initialize the full-rank DPP.

Comparing test log-likelihood values and training time is consistent with previous studies (Gillenwater et al. 2014; Mariet and Sra 2015), and allows a direct comparison of low-rank and full-rank DPPs, which is the focus of this paper.

Our experiments are based on two 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 (Gillenwater et al. 2014; Mariet and Sra 2015). In particular, (Gillenwater et al. 2014) 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.

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

2. **MS Store -** This is a proprietary dataset composed of shopping baskets purchased in Microsoft's Web-based 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. Recall from Section 2.1 that a low-rank DPP places zero probability mass on subsets with more than K items, where K is the number of trait dimensions. With this constraint in mind, we use $K = 15$ trait dimensions for the low-rank DPP models trained on this data, since the largest observed basket in this dataset is composed of 15 items.

Baby Registry

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

MS Store

	F-Rank	L-Rank
All Products	**-15.10**	-15.23

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

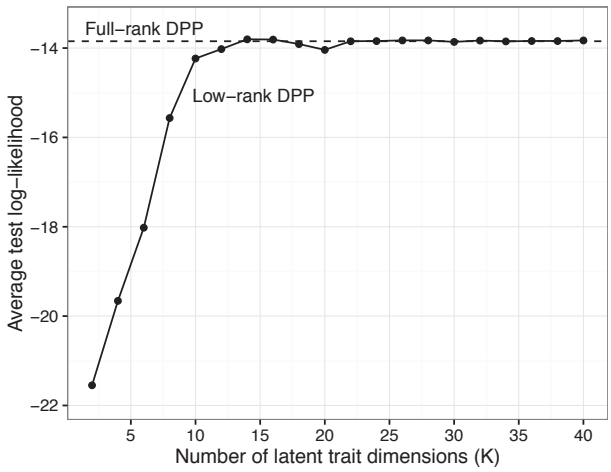


Figure 1: Average test log-likelihood of the low-rank DPP on the Amazon apparel dataset, as a function of the number of latent trait dimensions K. The average test log-likelihood of the full-rank DPP on this dataset is shown for comparison.

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

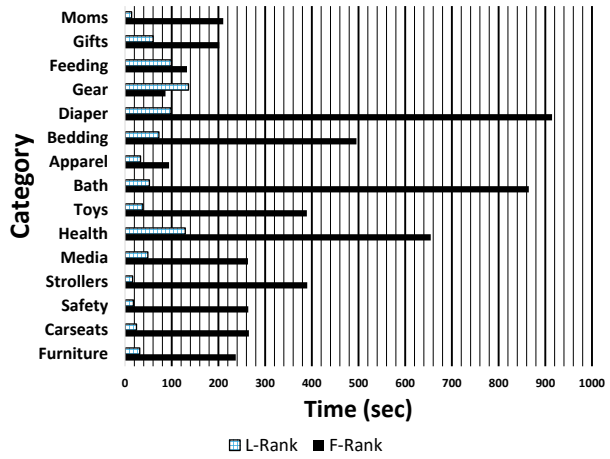
We begin with comparing test log-likelihood values of the low-rank DPP model presented in this paper with the full-rank DPP trained using Picard iteration. Table 1 depicts the average test log-likelihood values of both models across the different categories of the Baby Registries dataset as well as the 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. In general, the differences in the log-likelihood values are very small.

Figure 1 shows the average test log-likelihoods of the full-rank DPP and the low-rank DPP for the Amazon apparel dataset as a function of the number of low-rank DPP trait dimensions, K (apparel is one of the most popular item categories in the Amazon baby registry dataset). We see that the low-rank DPP reaches approximate parity with the full-rank DPP at $K = 14$, and there is no improvement for larger values of K. The largest observed basket in this dataset contains 21 items. It is important to note that for a K-rank DPP, one cannot sample a basket larger than K items. In practice this is not an issue for many applications, and a low-rank DPP can fully approximate the expressive power of a full-rank DPP when K is set to the size of the largest observed basket in the data (and sometimes for even smaller values of K, as is the case for the Amazon apparel dataset). Also, recall from Section 2.5 that for a K-rank DPP we can still compute predictions for a basket A, of any arbitrary size $|A|$, using the conditional matrix $\mathbf{V}^A$.

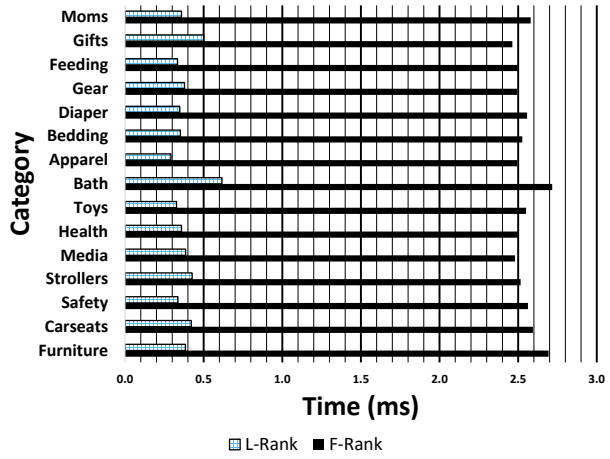
Training Time A key contribution of the Picard iteration method was the improvement of training time (convergence time) by up to an order of magnitude (Mariet and Sra 2015) compared to previous methods. However, the Picard iteration method requires inverting an $M \times M$ full-rank $(\mathbf{L} + \mathbf{I})$ matrix, where M is the number of items in the catalog. This matrix inversion operation has a $O(M^3)$ time complexity. As discussed in Section 2.3, the cost of training our low-rank model is far lower when $K \ll M$. This translates into considerably faster training times, particularity in cases where the item catalog is large.

Figure 2(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.

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) Training time, in seconds



(b) Average Prediction time per basket, in milliseconds

Figure 2: Training and prediction time of low rank DPP (L-Rank) vs. full rank DPP (F-Rank).

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 (Gillenwater et al. 2014; Mariet and Sra 2015) 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

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

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 2(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 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, we see that these prediction times are quite small. We see 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, 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 is only 83.6 ms per basket, which is 19.9x faster than the full-rank model.

Our low-rank DPP model also provides substantial savings in memory consumption 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.

4 Conclusions

In this paper we have presented a new method for learning the DPP kernel from observed data, using 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. Through an experimental evaluation using several real-world datasets in the domain of recommendations for shopping baskets, we have shown that our low-rank DPP model is substantially faster and more memory efficient than previous full-rank DPP approaches for both training and prediction.

Acknowledgements

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

References

Affandi, R. H.; Kulesza, A.; Fox, E.; and Taskar, B. 2013. Nystrom approximation for large-scale determinantal processes. In *AISTATS*, 85–98.

Affandi, R. H.; Fox, E.; Adams, R.; and Taskar, B. 2014. Learning the parameters of determinantal point process kernels. In *ICML*, 1224–1232.

Affandi, R. H.; Fox, E.; and Taskar, B. 2013. Approximate inference in continuous determinantal processes. In *NIPS*, 1430–1438.

Anari, N.; Oveis Gharan, S.; and Rezaei, A. 2016. Monte carlo markov chain algorithms for sampling strongly rayleigh distributions and determinantal point processes. In *29th Annual Conference on Learning Theory*, 103–115.

Chao, W.-L.; Gong, B.; Grauman, K.; and Sha, F. 2015. Large-margin determinantal point processes. In *UAI*.

Gillenwater, J. A.; Kulesza, A.; Fox, E.; and Taskar, B. 2014. Expectation-maximization for learning determinantal point processes. In *NIPS*, 3149–3157.

Gillenwater, J.; Kulesza, A.; and Taskar, B. 2012. Near-optimal map inference for determinantal point processes. In *NIPS*, 2735–2743.

Gillenwater, J. 2014. *Approximate inference for determinantal point processes*. Ph.D. Dissertation, University of Pennsylvania.

Kulesza, A., and Taskar, B. 2010. Structured determinantal point processes. In *NIPS*, 1171–1179.

Kulesza, A., and Taskar, B. 2011a. k-DPPs: Fixed-size determinantal point processes. In *ICML*, 1193–1200.

Kulesza, A., and Taskar, B. 2011b. Learning determinantal point processes. In *UAI*.

Kulesza, A., and Taskar, B. 2012. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning* 5(2-3):123–286.

Kwok, J. T., and Adams, R. P. 2012. Priors for diversity in generative latent variable models. In *NIPS*, 2996–3004.

Li, C.; Jegelka, S.; and Sra, S. 2016a. Efficient sampling for k-determinantal point processes. In *AISTATS*, 1328–1337.

Li, C.; Jegelka, S.; and Sra, S. 2016b. Fast dpp sampling for nyström with application to kernel methods. *arXiv preprint arXiv:1603.06052*.

Lin, H., and Bilmes, J. 2012. Learning mixtures of submodular shells with application to document summarization. In *UAI*.

Macchi, O. 1975. The coincidence approach to stochastic point processes. *Advances in Applied Probability* 83–122.

Mariet, Z., and Sra, S. 2015. Fixed-point algorithms for learning determinantal point processes. In *ICML*, 2389–2397.

Mariet, Z., and Sra, S. 2016a. Diversity networks. In *International Conference on Learning Representations*.

Mariet, Z., and Sra, S. 2016b. Kronecker determinantal point processes. *arXiv preprint arXiv:1605.08374*.

Nesterov, Y. 1983. A method of solving a convex programming problem with convergence rate O(1/sqr(k)). *Soviet Mathematics Doklady* 27:327–376.

Robbins, H., and Monro, S. 1951. A stochastic approximation method. *The Annals of Mathematical Statistics* 22(3):400–407.

Snoek, J.; Zemel, R.; and Adams, R. P. 2013. A determinantal point process latent variable model for inhibition in neural spiking data. In *NIPS*, 1932–1940.

Tschiatschek, S.; Djolonga, J.; and Krause, A. 2016. Learning probabilistic submodular diversity models via noise contrastive estimation. In *AISTATS*, 770–779.