6.883 Learning with Combinatorial Structure
Note for Lecture 22
Author: Sung Min (Sam) Park

Last time we introduced Determinantal Point Processes (DPP) as a probabilistic model of diversity. Recall that a point process is a probability distribution over subsets of a ground set. A DPP is a point process parameterized by a similarity matrix K or L.

We also saw that many inference problems on DPPs can be solved in polynomial time using basic routines in linear algebra (eigendecomposition, determinant, etc.)

In this lecture, we see a specific application of DPPs, some extensions to make it more efficient, and other connections.

1 Application: pose estimation

We describe an application of DPPs to solve the problem of pose estimation. We are given an image X, and our goal is to select or tag some subset of all possible locations $\mathcal{Y}(X)$ that we think contain persons, also possibly along with their pose (relative position and orientation of arms, legs).

The reason why we might want to use DPPs to model something like this is because we know intuitively that people are spatially spread out from each other; the presence of a person in one part of an image is negatively correlated with the presence of another person in the same part.

1.1 Model

We outline one approach to learning a DPP for pose estimation. We decompose the model as follows: each pose labeling is given both a quality and a diversity score. Then, the similarity score between two poses is given by:

$$L_{ij} = x_i^\top x_i = q_i \phi_i^\top \phi_j q_j$$

This defines the feature matrix $\Phi_{ij} = \phi_i^\top \phi_j$. Probability of sampling a set of poses S under the L-ensemble DPP is proportional to

$$\det(L_S) = \left(\prod_{i \in S} q_i^2 \right) \det(\Phi_S)$$

Going back to our geometric visualization of DPPs last time, quality score scales the length of the feature vector, and diversity score captures the orthogonality between two feature vectors. Higher scores increase the volume of the parallelepiped, thus increasing the probability that this pose labeling is chosen.

For the diversity feature ϕ, one can use something simple like location. For quality q, one can incorporate the likelihood score of an object detector (say for a body part). For the ith patch of the image, we may write $q_i(X) = \exp(\theta^T f_i(X))$ where θ are the weights of the detector and $f_i(X)$ are the features describing the ith patch.

This gives the parameterized likelihood:

$$p_\theta(Y|X) = \frac{\det(L_Y(X;\theta))}{\det(L(X;\theta)+I)}$$

1.2 Learning

We observe some images and detections in them as $\{(X^t, Y^t)\}_{t=1}^N$. We can just find the MLE using the log-likelihood, $\log \prod_{t=1}^N p_\theta(Y^t|X^t)$. Fortunately, this is concave in θ so we can run gradient ascent (see [1] for more details).

An example of applying DPPs to pose estimation is shown in Figure 1. We can see from the top right figure that though the initial feature detections are highly overlapping and concentrated in the center, we can recover the approximate pose of each person by inducing negative correlation using a DPP.

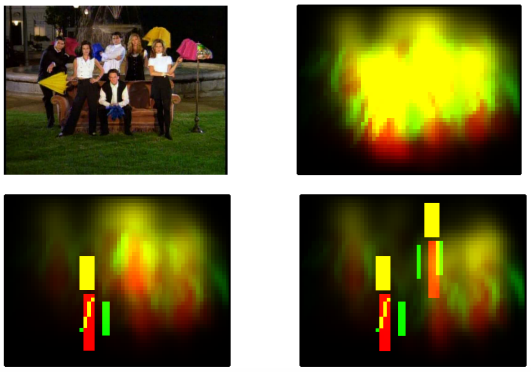

Figure 1: Pose estimation from an image [2]

There are other approaches to learning the DPPs from observations besides this quality–diversity decomposition, such as learning a weighted combination of kernel matrices or, going further, an entire PSD similarity matrix.

2 Extensions

While doing inference with DPPs only use basic linear algebra (eigendecomposition, determinant, etc.), this is not super cheap once the ground set (and the corresponding matrix) becomes very large. In our example above, each "item" is a configuration of parts, so the ground set could be exponentially large in the number of parts.

We show one technique for reducing the dimensionality of our DPP matrix below.

2.1 Dual representation

Before we were looking at $L = B^T B$ where B is the $n \times d$ matrix with one item per row (d is the number of features).

The idea of dual representation is to instead look at the matrix $C = BB^T$, which is $d \times d$ so will be smaller when $d < n$. At the end of the day, all we need is to compute the eigenvectors and eigenvalues of L; can we learn these from C?

Yes! The eigenvectors of C and L are related by the SVD of B (they correspond to left and right singular vectors).

Lemma 1. *If* $C = \sum_{i=1}^{d} \lambda_i \hat{v}_i \hat{v}_i^T$, *then* $L = \sum_{i=1}^{n} \lambda_i v_i v_i^T$ *where* $v_i = \frac{1}{\sqrt{\lambda_i}} B^T \hat{v}_i$.

Proof. The above decomposition follows from the fact that if v is an eigenvector of $C = BB^T$ with eigenvalue λ, then $B^T B(B^T v) = B^T(BB^T)v = B^T(\lambda v) = \lambda B^T v$, i.e. $B^T v$ is eigenvector of $L = B^T B$ with the same eigenvalue. □

From above, we can also easily compute $K = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_i + 1} v_i v_i^T$.[1]

2.1.1 Dual sampling

It remains to address how to sample from this new dual representation. We follow the same two step procedure from last time, where we first sample an elementary DPP P^T, then sample items using the chosen P^T.

Step 1 is same as before, where we sample a component P^T by sampling a subset T of eigenvectors: eigenvector i is sampled with probability $\frac{\lambda_i}{\lambda_i + 1}$. We can do this without

[1] Note that if $n > d$, $\lambda_i = 0$ for $i > d$.

directly working with v_i's since we just need to know the λ_i's, which we know from the eigendecomposition of C.

In Step 2, we sample item i with probability $P(i) = \frac{1}{|T|}\sum_{j \in T}(v_j^T \mathbb{1}_i)^2)$ as before. Again, the geometric intuition is that we are sampling proportional to the "height" of the ith feature vector.

But now we can rewrite

$$P(i) = \frac{1}{|T|}\sum_{j \in T}(\hat{v_j}^T b_i)^2$$

where b_i is the i^{th} column of B. Notice $P(i)$ can be computed using only the eigendecomposition of C.

Finally, we do two post-processing steps as before: i) project out the chosen element i from $\hat{V}$ (this is the space spanned by $\hat{v}_j, j \in T$); ii) orthonormalize $\hat{V}$ w.r.t the dot product $\langle \hat{v}_1, \hat{v}_2 \rangle = \hat{v}_1^T C \hat{v}_2^T$. Redefine T to range over the new eigenbasis of $\hat{V}$, and iterate to sample other items.

The time complexity of this sampling procedure is $O(ndk^2 + d^2k^3)$. For more details, see Algorithm 3 in [1].

2.1.2 Other techniques to reduce dimension

If d is large, we can also use random projection methods based on classical ideas such as Johnson-Lindenstrauss projections to sketch the feature space and reduce the dimension.

A special case when our output is structured is called structured-DPPs. In that case, the ground set may be doubly exponentially large. Fortunately, one can combine the above dual representation idea with a factorization over the parts (e.g. graphical models) and apply message-passing algorithms to do efficient inference. See [1] for more details.

2.2 Exact cardinality constraint

Depending on the application, we may want to sample exactly k items for some k. We call this a k-DPP. For example, if we are running a search engine, we may want to return a set of diverse responses of fixed size to the user.

We see how to compute the probability of sampling y conditioned on the event that we sample a set of size k:

$$P_L(Y|\text{set of size } k) = \frac{\det(L_Y)}{\sum\limits_{\substack{|Y'|=k \\ Y' \subseteq \mathcal{V}}} \det(L_{Y'})} \mathbb{1}[|Y| = k]$$

We can simplify the normalizing constant as follows:

$$\begin{aligned}
\sum_{\substack{|Y'|=k \\ Y' \subseteq \mathcal{V}}} \det(L_{Y'}) &= \sum_{\substack{|Y'|=k \\ Y' \subseteq \mathcal{V}}} \det(L+I) P_L(Y') \\
&= \sum_{\substack{|Y'|=k \\ Y' \subseteq \mathcal{V}}} \sum_{T \subseteq \{1,\cdots,n\}} P^T(Y') \prod_{i \in T} \lambda_i && \text{using the elementary DPP decomposition} \\
&= \sum_{|T|=k} \sum_{\substack{|Y'|=k \\ Y' \subseteq \mathcal{V}}} P^T(Y') \prod_{i \in T} \lambda_i && \text{noticing } P^T(Y') = 0 \text{ unless } |T| = k \\
&= \sum_{|T|=k} \prod_{i \in T} \lambda_i && \text{since } P^T \text{ always returns a set of size } k \\
&= e_k(\lambda_1, ..., \lambda_n)
\end{aligned}$$

where the last expression is the kth elementary symmetric polynomial, and can be computed efficiently using recursion or dynamic programming.

2.2.1 Sampling from a k-DPP

Now we modify Step 1 of the original sampling procedure since the weights of the elementary DPPs are now different:

$$P_L^k = \sum_{|T|=k} \frac{1}{e_k^n} \prod_{i \in T} \lambda_i P^T$$

Computing the constant here naively would be exponential, but one can again take advantage of the elementary symmetric polynomials to get an efficient sampling procedure that iterates over all n eigenvectors independently; see Algorithm 8 and Theorem 5.2 in [1] if you are curious for more details.

Step 2 is same as before once we have sampled the subset of eigenvectors (and note that we sample exactly k items since our subset is of size k)

2.3 Concentration

Lastly, we note that DPPs have good concentration properties. Intuitively, this is because in order for a significant deviation to occur, a large fraction of the sample have to deviate in roughly the same direction; however, this is precisely what is penalized by DPPs, so deviations are unlikely in DPPs.

3 Connections to Spanning trees

We briefly introduce a connection of DPPs to uniformly sampling spanning trees[2] from an undirected graph $G(V, E)$ where we denote $|E| = m$, $|V| = n$. In fact, uniformly sampling a spanning tree is in fact a DPP for some choice of matrix M. We will see shortly what this matrix M is, but to get a little intuition, we see that M has to depend on the connectivity of the graph. For instance, consider two complete graphs connected by an edge. We know that the edge connecting the two components has to be present in every tree; removing this edge would disconnect the graph. Clearly, there is some measure of connectivity or importance of an edge at work here.

Define a feature vector for edge $e = (u, v)$ as

$$b_e(i) = \begin{cases} 1 & \text{if } i = u \\ -1 & \text{if } i = v \\ 0 & \text{if otherwise} \end{cases}$$

Then, let B be the $m \times n$ matrix with $b_e(i)$ as rows of the matrix.

The *Laplacian* of the graph is the $n \times n$ matrix $L = B^T B$. It can also be written as $D - A$ where D is the degree matrix and A is the adjacency matrix. The following theorem shows that L captures some information about the spanning trees of G.

Theorem 1 (Matrix-Tree Theorem). $\frac{1}{n} \det(L + \frac{1}{n} \mathbb{1}\mathbb{1}^T)$ = # *of spanning trees of* $G = |\mathcal{T}|$

(Note that L by itself is not full rank since $\vec{1}$ is in its kernel).

However, L is not quite directly useful for sampling. For that, we define the *transfer-current* matrix as follows. Define $y_e = (L^{\dagger\frac{1}{2}})b_e$ where $L^{\dagger\frac{1}{2}} = (L^\dagger)^{\frac{1}{2}}$. Then, define a new matrix $\tilde{B} = BL^{\dagger\frac{1}{2}}$, and then $M = \tilde{B}\tilde{B}^T = BL^\dagger B$. Note $M_{e,f} = \langle y_e, y_f \rangle$.

[2] This can be generalized to some other matroids (*regular* matroids), where we are sampling a random basis from a matroid

3.1 Properties

We note a few properties of the DPP generated by M. First, note that edges are negatively correlated, as they should. It remains to show that we in fact get a uniform distribution over all spanning trees by sampling from this DPP.

The following lemma shows that M is an elementary DPP by itself.

Lemma 2 (Spielman-Srivastava [3])**.** *M is a projection matrix, i.e. has eigenvalues 1 and 0 with multiplicity $n-1$ and $m-(n-1)$, respectively.*

In fact, if T is the subset of edges sampled from our DPP, $\mathbb{E}[|T|] = n-1$. In fact, T has to be a spanning tree since if T were a cycle, the corresponding vectors would be linearly dependent, and hence have zero volume and will be never sampled.

For any tree $F \subseteq E$,

$$P(F) = \det(\tilde{B}_F \tilde{B}_F^T) = \det(L + \frac{1}{n}\mathbb{1}\mathbb{1}^T)^{-1} \cdot \det(B_F B_F^T) = \frac{1}{n|\mathcal{T}|} \cdot n = \frac{1}{|\mathcal{T}|}$$

so we do get a uniform distribution.[3]

The probability of sampling a particular edge e is

$$P(e \in T) = \det(M_e) = b_e^T L^\dagger b_e$$

which is also known as the *effective resistance* of edge e.

We give two interpretations of this quantity:

- **Electrical resistance**: If we view the graph as an electrical network where the edges are resistors, then effective resistance of edge $e = (u, v)$ is the potential difference across edge e if we send 1 unit of electrical flow from u to v.
- **Random walks**: The commute time $C(u, v)$ is the expected number of steps for a random walk starting at u to reach v at least once and then return to u. It was shown in [4] that $C(u, v) = \text{Reff}(u, v) \cdot 2m$.

  We also remark that there is a remarkably simple algorithm based on random walks that allows uniformly sampling a spanning tree.

In a rough sense, the effective resistance of an edge measures how "important" the edge is. Unsurprisingly, this quantity also shows up in problems like graph sparsification, where one wants to preserve connectivity properties of a graph by sampling edges.

[3] Note that the second to last equality is non-trivial and requires some work.

References

[1] A. Kulesza and B. Taskar, *Determinantal point processes for machine learning*, vol. 5. Foundations and Trends in Machine Learning, 2012.

[2] A. Kulesza and B. Taskar, "Structured determinantal point processes," in *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[3] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1913–1926, 2011.

[4] A. Chandra, P. Raghavan, W. Ruzzo, R. Smolensky, and P. Tiwari, "The electrical resistance of a graph captures its commute and cover times," in *Symposium on Theory of Computing (STOC)*, 1989.