Clustering via Determinantal Point Processes

Paul Vernaza, Jiaji Zhou, Kris Kitani, J.Andrew Bagnell

January 11, 2014

1 Recap on DPPs

Determinantal Point Processes (DPPs)[1] is a probability measure on all subsets of a ground discrete set $G = \{1, ..., N\}$. Suppose we draw a random subset $\mathbf{Y}$, then we have for every $A \subseteq G$, the probablity of A is contained in $\mathbf{Y}$ is equal to the determinant of the submatrix indexed by the marginal kernel matrix K:

$$\mathcal{P}(A \subseteq \mathbf{Y}) = \det(K_A) \tag{1}$$

Note that the eigenvalues of K must be nonnegative and bounded by one so as to specify the inclusion probablities. It is convenient, however, to directly assign atomic probability up to proportionality. Given an *L-ensemble*, whose element measures the pairwise similarity between item feature vectors, we have:

$$P_L(\mathbf{Y} = Y) \propto \det(L_Y) \tag{2}$$

The normalization constant and log partition would be:

$$Z = \sum_{\mathbf{Y} \subseteq V} \det(L_Y) = \det(L + I) \tag{3}$$

$$\log(Z) = \log(\det(L + I)) \tag{4}$$

The benefit of using *L-ensemble* L instead of marginal kernel K is that it only needs to be positive semidefinite with no restriction that the eigen values have to be bounded by one. Therefore, given item feature vectors ϕ_i, we can use various kernel similarity matrix for L. Note than $L + I$ is also a valid kernel matrix.

$$L = \begin{pmatrix} k(\phi_1, \phi_1) & k(\phi_1, \phi_2) & \cdots \\ k(\phi_2, \phi_1) & k(\phi_2, \phi_2) & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

1.1 Eigen Decomposition for Feature Representation

Given a kernel matrix for *L-ensemble* L, since L is positive semi-definite, we can apply eigen decomposition on L to get low dimensional feature embedding in the kernel space:

$$L = Q\Lambda Q^T = Q\Lambda^{1/2}\Lambda^{1/2^T}Q^T = V^T V \tag{5}$$

, where each column v_i of V corresponds to feature vector that lies in a low dimensional embedding. Note that the linear inner product between these feature vectors is equal to the kernel product of orginal feature vectors. $L_{ij} = k(\phi_i, \phi_j) = v_i^T v_j$.

2 Submodularity of Log Determinant

Determinants also have a nice geometrical interpretation. Let $L = V^T V$ and denote columns of V by v_i, then:

$$\det(L_Y) = \mathrm{Vol}^2(\{v_i\}_{i\in Y}) \tag{6}$$

, where Vol^2 denotes the squared volumn of the parallelpiped spanned by the input feature vectors. We now show that $\log\det$ is submodular: The function $\log\det(\cdot)$ is submodular iff.

$$\begin{aligned}&\log\det(v_1,\ldots,v_k,x) - \log\det(v_1,\ldots,v_k)\\ &\quad\geq \log\det(v_1,\ldots,v_k,v_{k+1},x) - \log\det(v_1,\ldots,v_k,v_{k+1}),\end{aligned} \tag{7}$$

for all $v_i \in \bigcup_j\{\phi_j\}$, which holds iff.

$$\frac{\mathrm{Vol}(v_1,\ldots,v_k,v_{k+1},x)}{\mathrm{Vol}(v_1,\ldots,v_k,v_{k+1})} \leq \frac{\mathrm{Vol}(v_1,\ldots,v_k,x)}{\mathrm{Vol}(v_1,\ldots,v_k)}. \tag{8}$$

Consider writing $v_{k+1} = v_{k+1}^{\perp} + v_{k+1}^{\parallel}$, where $v_{k+1}^{\perp} \perp \mathrm{span}\{v_1,\ldots,v_k,x\}$ and $v_{k+1}^{\perp} \perp v_{k+1}^{\parallel}$. Note that (8) holds with equality if $v_{k+1} = v_{k+1}^{\perp}/\|v_{k+1}^{\perp}\|$, since parallelpiped volume is invariant with respect to unit extrusion in an orthogonal direction. Together with the multilinearity of determinants, we can therefore write the LHS of (8) as

$$\frac{\|v_{k+1}^{\perp}\|\,\mathrm{Vol}(v_1,\ldots,v_k,\frac{v_{k+1}^{\perp}}{\|v_{k+1}^{\perp}\|},x) + \mathrm{Vol}(v_1,\ldots,v_k,v_{k+1}^{\parallel},x)}{\|v_{k+1}^{\perp}\|\,\mathrm{Vol}(v_1,\ldots,v_k,\frac{v_{k+1}^{\perp}}{\|v_{k+1}^{\perp}\|}) + \mathrm{Vol}(v_1,\ldots,v_k,v_{k+1}^{\parallel})}. \tag{9}$$

Since $v_{k+1}^{\parallel} \in \text{span}\{v_1, \ldots, v_k, x\}$, $\text{Vol}(v_1, \ldots, v_k, v_{k+1}^{\parallel}, x) = 0$. We therefore have

$$\begin{aligned}
\frac{\text{Vol}(v_1, \ldots, v_k, v_{k+1}, x)}{\text{Vol}(v_1, \ldots, v_k, v_{k+1})} &= \frac{\text{Vol}(v_1, \ldots, v_k, \frac{v_{k+1}^{\perp}}{\|v_{k+1}^{\perp}\|}, x)}{\text{Vol}(v_1, \ldots, v_k, \frac{v_{k+1}^{\perp}}{\|v_{k+1}^{\perp}\|}) + \frac{1}{\|v_{k+1}^{\perp}\|}\text{Vol}(v_1, \ldots, v_k, v_{k+1}^{\parallel})} \\
&= \frac{\text{Vol}(v_1, \ldots, v_k, x)}{\text{Vol}(v_1, \ldots, v_k) + \frac{1}{\|v_{k+1}^{\perp}\|}\text{Vol}(v_1, \ldots, v_k, v_{k+1}^{\parallel})} \\
&\leq \frac{\text{Vol}(v_1, \ldots, v_k, x)}{\text{Vol}(v_1, \ldots, v_k)}.
\end{aligned}$$

This proves that $\log\det$ is submodular.

3 DPPs for clustering

DPPs model the diversity of subsets. For clustering, we want the opposite, i.e., points that belong to the same cluster should be concentrated. One good measurement of dispersity for a cluster is the log partition value.

Therefore we seek to minimize the sum of log determinant of each submatrix that corresponds to partitions of data into clusters. Suppose we want to partition the data into k clusters, the optimization objective would be:

$$\begin{aligned}
\min_{S_1,\ldots,S_k} f(S) = \sum_{i=1}^{k} g(S_i) = \sum_{i=1}^{k} \log\det(L_{S_i}) \\
\cap_{i=1}^{k} S_i = \emptyset, \quad \cup_{i=1}^{k} S_i = G
\end{aligned} \tag{10}$$

Note that $g(\cdot) = \log\det(\cdot)$ is submodular. The greedy splitting algorithm[3] is guaranteed an upper bound of $(2 - \frac{2}{k}) * OPT$.

3.1 Queyranne's algorithm for 2-way Partition

If we let $k = 2$ in (10), which is to partition the data into 2 clusters, we can rewrite the problem as

$$\begin{aligned}
\min_{S \subset G} f(S) = g(S) + g(G/S) \\
0 < |S| < |G|
\end{aligned} \tag{11}$$

The objective is symmetric and submodular, and note if we do not add the cardinality constraint, the empty set or the full ground set will be returned as the trival solutions. Queyranne's algorithm[2] can be used to find an non-trival optimal solution in $O(JN^3)$, where J is the cost expended per evaluation of the submodular function (in this case, $\log\det(\cdot)$). A naive implementation of $\log\det$ yields $J = O(N^3)$, resulting in a total

worst-case runtime of $O(N^6)$. However, this can be reduced to $O(N^4)$ via the method described here.

First, we review Queyranne's algorithm. The algorithm conceptually proceeds as follows.

1. Choose a pair of elements $(s, t) \in V \times V$
2. Find the optimal partition of V separating s and t
3. Find the optimal partition of V not separating s and t
4. Return the better of these two partitions

Step 3 may be performed recursively by applying the algorithm to a new set where s and t are merged into a single element, defining a new objective function in the natural way. The algorithm terminates in $|V| = N$ recursive calls. The difficult part of the above is finding a pair of elements such that we can efficiently find the optimal partition separating them. Queyranne's algorithm accomplishes this by finding a *pendent pair*: (s, t) is called a pendent pair iff. the optimal partition separating (s, t), isolates t. Such a pair can be found using $O(N^2)$ calls to the submodular function, yielding a net $O(N^3)$ runtime, neglecting the cost of calling the submodular function.

The pendent pair algorithm orders the input set in such a way that the penultimate element is s and the last element is t in a pendent pair (s, t). The algorithm proceeds as follows

1. Input: a set of ordered elements W and a set of yet-unordered elements Q
2. Sort the elements of $q \in Q$ by the value of the function $\text{key}(q) = \log\det(q \bigcup W) - \log\det q$
3. Append to W and delete from Q the least element determined in this way
4. Recurse

The ordering is given by the order in which W was costructed. The algorithm terminates in a number of recursive calls equal to the size of the input set K. Each recursive call performs $O(K)$ calls to $\log\text{vol}$. Since $K \leq N$, the pendent pair algorithm calls $\log\text{vol}$ a number of times bounded by $O(N^2)$.

3.1.1 Optimizing pendent-pair-finding

We can optimize pendent-pair-finding for the $\log\text{vol}$ objective by incrementally computing the required determinants. Note that pendent-pair in our case is called with a set of parallelepipeds. The algorithm works by storing along with each parallelepiped a corresponding "projected" parallelepiped. The algorithm maintains the invariant that the projected parallelepipeds in Q stay orthogonal to all of the parallelepipeds in W. Given this invariant, the quantity $\log\text{vol}(q \bigcup W)$ in $\text{key}(q)$ is equal to $\log\text{vol}\, q' + \log\text{vol}\, W$,

where q' is the current projected version of q. Since $\log \operatorname{vol} W$ is a constant wrt. q in $\text{key}(q)$, it need not be computed, and we need only to compute the volume of the usually smaller parallelepiped q'. The $\log \operatorname{vol} q$ component of $\text{key}(q)$ may be computed and cached at the point in the outer loop of Queyranne's algorithm at which q is created by merging smaller parallelepipeds.

3.1.2 Worst-case complexity analysis

First, note that the $\log \operatorname{vol} q$ factor may be computed at a cost of $O(N^3)$ in the outer loop of Queyranne's algorithm. The complexity of pendent-pair is determined by the cost of maintaining the loop (orthogonality) invariant and the cost of computing the determinants of the projected parallelepipeds.

Suppose pendent-pair is called with a set of K parallelepipeds. Denote by s_i the number of vectors in the ith such parallelepiped. The orthogonality invariant is maintained by, upon choosing a new element q of W from Q, making all elements of Q orthogonal to q. The total cost of doing so is therefore bounded by the cost of making every parallelepiped orthogonal to every other parallelepiped (and itself), which is equal to $\sum_{i,j=1}^{K} N s_i s_j$ (making i orthogonal to j involves $s_i s_j$ projections costing $O(N)$ each).

The cost of computing the determinant of the projected parallelepiped q_i' is bounded by the cost of computing s_i^2 projections, each at a cost of N. The total cost over all K recursive calls of pendent-pair is therefore bounded by $K \sum_{i=1}^{K} N s_i^2$. Summing this with the projection cost, we obtain the total worst-case running time bound of $K \sum_i N s_i^2 + \sum_{i,j} N s_i s_j$ for pendent-pair.

We can therefore bound the total worst-case running time of pendent-pair by solving a quadratic optimization problem over vectors of possible parallelpiped sizes $s \in \{0, \ldots, N\}^K$. By writing $Q = 11^T + KI$, the optimization problem can be written as

$$\max_s N s^T Q s \text{ subject to } 1^T s = N. \tag{12}$$

By relaxing the integrality constraint entirely, we can exactly compute an upper bound on this quantity. Relaxing and solving the QP yields a bound of

$$\frac{N^3}{1^T Q^{-1} 1}. \tag{13}$$

The denominator can be obtained by observing that 1 is an eigenvector of Q and therefore of Q^{-1} as well. The corresponding eigenvalue of Q is equal to $2K$, which is transformed to $(2K)^{-1}$ in Q^{-1}. The denominator is therefore equal to $\frac{1}{2}$, yielding a final bound of $2N^3$ for the complexity of pendent-pair.

3.2 Greedy Splitting for k-way Partition

Greedy Splitting algorithm[3] works by recursively spliting one of the clusters into two clusters. Note that each round, by a heap structure for caching, one only have to consider the newly splitted two clusters.

1. Find the optimal 2-way split $(A_G, G \backslash A)$ for ground set G, $y_G = \min_{\emptyset \subset A_G \subset G}(f(A_G) + f(G \setminus A_G) - f(G))$; initialize the min-heap with (G, A_G, y_G); initialize the solution set $S = \{G\}$

2. Let (C, A_G, y_G) = min-heap.pop(); eliminate C from S and add A_G and $C \setminus A_G$ into S.

3. Split C_1 and C_2 into two clusters and record the value keys $y_{C_1} = \min_{\emptyset \subset A_{C_1} \subset C_1}(f(A_{C_1}) + f(C_1 \setminus A_{C_1}) - f(C_1))$, $y_{C_2} = \min_{\emptyset \subset A_{C_2} \subset C_2}(f(A_{C_2}) + f(C_2 \setminus A_{C_2}) - f(C_2))$; push (C_1, A_{C_1}, y_{C_1}) and (C_2, A_{C_2}, y_{C_2}) into the min-heap;

4. Jump to step 2 if $|S| < k$, else output S.

The total number of 2-way splitting would be $2k - 3$, so the computational complexity is $O(kN^4)$.

4 Preliminary Experimental Result

References

[1] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *arXiv preprint arXiv:1207.6083*, 2012.

[2] Maurice Queyranne. Minimizing symmetric submodular functions. *Mathematical Programming*, 82(1-2):3–12, 1998.

[3] Liang Zhao, Hiroshi Nagamochi, and Toshihide Ibaraki. Greedy splitting algorithms for approximating multiway partition problems. *Mathematical Programming*, 102(1):167–183, 2005.