**CS 329M: Topics in Artificial Intelligence** **Spring 2017**

Lecture 3: April 17

*Lecturer: James Zou* *Scribe: Vighnesh Sachidananda*

## 3.1 Tensor Decomposition

In this section, we quickly recap the goal of tensor decomposition problem we introduced in the last lecture. Then we revisit an efficient algorithm to compute the decomposition as defined. Lastly, we expound on applications of Matrix and Tensor Completion.

### 3.1.1 Recap

Recall our notation of tensor decomposition of a tensor $T$ into an approximation by the sum of many low rank tensors

$$T \approx \sum_r \alpha_r u_r \otimes v_r \otimes w_r \tag{3.1}$$

From this definition we see that the low rank tensors are composed as the product of vectors $u_r, v_r$ and $w_r$. This is analogous to our two-dimensional matrix case (which is the sum of low rank matrices).

We pause to recognize that some examples of this data structure exists in images (where the RGB channels define the z-axis of the tensor) and in biomedical datasets (where we can align (1) patient, (2) diagnosis, and (3) treatment as axes of our tensor).

Additionally, we notice that "collapsing" the tensor into a matrix allows us to use methods we may be more familiar with but loses some of the inherent structure present in the tensor model.

### 3.1.2 Jennrich's Algorithm

Given a tensor $T$, we seek a decomposition into the parameters of the model shown in equation 3.1. Jennrich's Algorithm achieves this goal. As a reminder, the algorithm is as follows

1. Generate random vectors $a$ and $b$, we take $T_a \stackrel{\text{def}}{=} T(\cdot, \cdot, a)$ and $T_b \stackrel{\text{def}}{=} T(\cdot, \cdot, b)$
2. Solve for $\{U_r\}$ which is the eigenvectors of $T_a T_b^+$ where $T_a T_b^+ = U\text{Diag}(\frac{a}{b})U^+$
3. Similarly solve for $\{V_r\}$ which is the eigenvectors of $T_b T_a^+$
4. Given $\{U_r\}$ and $\{V_r\}$ we are left to solve for $\{W_r\}$. For this we can simply solve the linear system for $\{W_r\}$.
5. Since $T$ is approximately low-rank, we can fix 2 of $\{U_r\}$, $\{V_r\}$, and $\{W_r\}$ and solve for the 3rd until convergence.

### 3.1.3 Matrix/Tensor Completion

The Matrix Completion problem gained fame from the Netflix Challenge. Given a matrix of Users × Items, where cells are populated by ratings, we seek to fill in the rest of the matrix (i.e. the ratings not observed by the algorithm). It is easy to see that matrix factorization can solve for this problem in this case and when presented with a tensor of incomplete entries, we can complete the tensor.

One optimization towards this goal is shown below

$$\min_{u,v,w,\alpha} \sum_{i,j,k \text{ observed}} \left( T[i,j,k] - \sum_r \alpha_r u_r \otimes v_r \otimes w_r[i,j,k] \right)^2 \tag{3.2}$$

## 3.2 Spectral Clustering

In this part we analyze how spectral methods can be used to uncover clusters of similar data points efficiently in a graph. First we introduce Graph Laplacian and then define Spectral Clustering as a relaxation of an optimization problem that relies on the Graph Laplacian.

### 3.2.1 Basic Graph Theory

We define a graph $G = (V, E)$, as a given set of vertices ($V$) and edges ($E$) between them. Furthermore, we defined edges in our case to be undirected and weighted. The weight we place on our edges is denoted as $w_{ij}$ which refers to the weight between vertices $V_i$ and $V_j$. Here our weights represent distance and low weights correspond to points that are "similar".

Our goal is to cluster our vertices (which can be viewed here as general data points) based on a notion of similarity as described above. In the case of a graph, the clustering can be viewed as taking a cut and the result can be viewed as partition of nodes in our graph. We are given the total number of partitions $k$ and seek to partition $V$ into $k$ sets $\{A_1, ..., A_k\}$.

Given two clustered sets $A$ and $B$, the cost of cut is the following

$$w(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

If we include a term that seeks to balance the size of the clusters generated, *RatioCut* is defined as follows

$$RatioCut(A_1, ..., A_k) = \sum_{i=1}^{k} \frac{w(A_i, \overline{A_i})}{|A_i|} \tag{3.3}$$

Note that the numerator represents the weight between $A_i$ and its complement. Furthermore, by dividing the term by $|A_i|$, the number of vertices in the cluster, cost is normalized as a function of the size of the cluster.

Our goal then becomes

$$\min_{A_1,...,A_k} \sum_{i=1}^{k} \frac{w(A_i, \overline{A_i})}{|A_i|}$$

### 3.2.2 Graph Laplacian

A choice of partition across $n$ vertices into $k$ clusters can be represented by a matrix $H \in \mathbb{R}^{n\times k}$. Each row of $H$ contains exactly one nonzero element since a vertex can only belong to one cluster. We have the following rule to fill matrix $H$

$$h_{ij} = \begin{cases} \frac{1}{\sqrt{|A_j|}} & \text{if } V_i \in A_j \\ 0 & \text{otherwise} \end{cases}$$

We will use Graph Laplacian to construct a solution to our minimization problem. We define the Graph Laplacian $L$ as

$$L = \underbrace{\text{Diag}(\sum_j w_{ij})}_{\text{Degree Matrix}} - \underbrace{W}_{\text{Adjacency Matrix}} \tag{3.4}$$

**Claim 3.1** *Given a vector $f \in \mathbb{R}^n$, we have*

$$f^T L f = \frac{1}{2}\sum_{i,j\in V} w_{ij}(f_i - f_j)^2$$

**Proof:**

$$\begin{aligned} f^T L f &= f^T D f - f^T W f \\ &= \sum_{i\in V} d_i f_i^2 - \sum_{i,j} w_{ij} f_i f_j \\ &= \frac{1}{2}\Big(\sum_{i\in V} d_i f_i^2 - 2\sum_{i,j} w_{ij} f_i f_j + \sum_{j\in V} d_j f_j^2\Big) \\ &= \frac{1}{2}\sum_{i,j\in V} w_{ij}(f_i - f_j)^2 \end{aligned}$$

■

Given this formulation, we can take each column of our matrix $H$, denoted as $h_j \in \mathbb{R}^n$ and expect the following result

$$\begin{aligned} h_j^T L h_j &= \frac{1}{2}\sum_{i,i'\in V} w_{ii'}(h_{ij} - h_{i'j})^2 \\ &= \sum_{i\in A_j, i'\notin A_j} \frac{w_{ii'}}{|A_j|} = \frac{w(A_j, \overline{A_j})}{|A_j|} \end{aligned}$$

This way we have shown that

$$\begin{aligned} &\min_{A_1,\dots,A_k} \; RatioCut = \sum_{j=1}^{k} \frac{w(A_j, \overline{A_j})}{|A_j|} \\ \iff &\min_{h_1,\dots,h_k} \; \sum_{j=1}^{k} h_j^T L h_j \\ &\text{s.t.} \quad H^T H = I \\ &\qquad h_j \in \left\{0, \frac{1}{\sqrt{|A_j|}}\right\} \; \forall j \in [k] \end{aligned}$$

### 3.2.3 Spectral Clustering

The issue we now face is that finding the minimum partition vectors $h_1, ..., h_k$ is NP-Hard. We will use a relaxation of our original problem to efficiently find the partitions. Dropping the last constraint, we end up with

$$\min_{h_1,...,h_k} \sum_{j=1}^{k} h_j^T L h_j$$

$$\text{s.t. } H^T H = I$$

This reduces to eigen-decomposition problem. By extracting the $k$ eigenvectors corresponding to the $k$ smallest eigenvalues in our Graph Laplacian $L$, we are to recover the cluster membership.

This relaxation defines the *Spectral Clustering algorithm* as follows

1. Generate Laplacian $L$ as defined in Equation 3.4
2. Compute $k$ smallest eigenvalues of $L$ and stack the corresponding eigenvectors as columns to obtain the matrix $H \in \mathbb{R}^{n \times k}$.
3. Now that each data point is represented as a $k$ dimensional point in space, we can run k-means clustering on $H$ and obtain k clusters as our output of the algorithm.