CS281B/Stat241B: Advanced Topics in Learning & Decision Making

Spectral Clustering

Lecturer: Michael I. Jordan *Scribe: David Sontag*

Wednesday, March 31, 2004.

1 Algorithm

Let $s_1, \ldots, s_n \in \mathbb{R}^l$ be data points that we wish to cluster. Define the n x n *affinity matrix* W by (e.g.):

$$W_{ij} = \exp(-\frac{1}{2\sigma^2}||s_i - s_j||^2).$$

This matrix is symmetric, nonnegative and positive semidefinite. In general, affinity matrices are symmetric and nonnegative, but not necessarily positive semidefinite.

Now define $D = \mathrm{diag}(W\mathbf{1})$, the matrix with the row sums of W along its diagonal, and let $L = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$. The clustering algorithm is as follows:

1. Let $u_1, u_2, \ldots, u_k$ denote the k largest eigenvectors of L, and let $X = (u_1, u_2, \ldots, u_k)$ be the n x k matrix formed with column i being the vector u_i. Think of the n rows of X as a new representation of the original n data points.
2. Re-normalize the rows of X to have unit norm, calling the resulting matrix Y (i.e. $Y_{ij} = \frac{X_{ij}}{(\sum_j X_{ij}^2)^{1/2}}$).
3. Cluster the rows of Y into k clusters using K-means. Assign data point s_i to row i's cluster.

Notice the similarity to Kernel PCA in step 1. The two would be equivalent if we didn't normalize by $D^{-1/2}$, i.e., if $L = W$.

2 Analysis in "ideal" case

Why does this algorithm work? To get some intuition we'll look at the "ideal" case, when the distance between clusters is larger than the distance of the data points within each cluster. In particular, we'll assume that the data points form k clusters that are infinitely far from each other. Ordering the data points by their cluster affiliation, we see that for $k = 3$, W decomposes into:

$$\hat{W} = \begin{bmatrix} \hat{W}^{(1)} & 0 & 0 \\ 0 & \hat{W}^{(2)} & 0 \\ 0 & 0 & \hat{W}^{(3)} \end{bmatrix}. \tag{1}$$

Pre- and post- multiplying by $D^{-1/2}$ does not change the block structure, so:

$$\hat{L} = \begin{bmatrix} \hat{L}^{(1)} & 0 & 0 \\ 0 & \hat{L}^{(2)} & 0 \\ 0 & 0 & \hat{L}^{(3)} \end{bmatrix}. \tag{2}$$

Let's walk through each step of the algorithm, using the above matrices $\hat{W}$ and $\hat{L}$ in place of W and L.

1. What are $\hat{L}$'s eigenvectors? Its eigenvalues are the union of the eigenvalues of each $\hat{L}^{(i)}$ matrix. The eigenvectors are those of $\hat{L}^{(i)}$'s, with appropriate padding of 0's. It can be shown that each $\hat{L}^{(i)}$ matrix has an eigenvalue of 1, and the next largest eigenvalue is strictly less than 1 (there is a gap).

   To find X, pick the largest eigenvector from each of the $\hat{L}^{(i)}$ matrices. This suffices, since each of these eigenvectors have corresponding eigenvalue 1, and all other eigenvalues are strictly less than 1. Don't get tricked into thinking that these eigenvectors are unique! Since 1 is a repeated eigenvalue of $\hat{L}$, these three eigenvectors span a *subspace*. Any rotation of $\hat{X}$ will be equally valid; that is, $\hat{X}$ can be replaced by $\hat{X}R$ for any orthogonal matrix $R \in \mathbb{R}^{3\times 3}$ ($R^T R = RR^T = I$).

2. The $\hat{Y}$ matrix now looks particularly simple:

$$\hat{Y} = \begin{bmatrix} \vec{1} & \vec{0} & \vec{0} \\ \vec{0} & \vec{1} & \vec{0} \\ \vec{0} & \vec{0} & \vec{1} \end{bmatrix} R \tag{3}$$

   for some rotation matrix R.

3. The rows of $\hat{Y}$ correspond to three orthogonal points lying on a unit sphere. Running k-means will immediately find the clusters.

An area of matrix analysis called matrix perturbation theory can now be applied for the setting when the 0's in the $\hat{W}$ affinity matrix are no longer zero (see Stewart & Sun, *Matrix Perturbation Theory*). If you bound the Frobenius norm of the difference between the two matrices by ϵ, the clusters are perturbed in some bounded way from their 90° orientations. As you perturb a matrix, its eigenvectors can change radically. However, the eigenvalues and subspaces of eigenvectors are more stable, providing some justification for why this algorithm works in the non-ideal case.

3 Normalized Cut

Another way to view the above algorithm is as a relaxed solution to an NP-hard optimization problem, called *normalized cut*. Assume that you have a complete graph $G = \langle V, E \rangle$ where you have one vertex for each of the n data points. The weight on the edges of the graph represent how 'similar' one data point is to another (and could be zero). In the Min-Cut problem, you try to find a set of edges of minimal weight, such that when you remove those edges, the graph becomes disconnected. Min-Cut is easily solved in polynomial time. However, min-cut does not take into consideration the size of the subgraphs that it tries to separate. From a clustering of data points perspective, it would be nice to be able to find disjoint clusters such that all points within each cluster are very similar, and all points external to a cluster are not similar.

In the *Normalized Cut* problem, we try to find k disjoint sets of vertices $(A_r)_{r=1,\ldots,k}$, that minimize:

$$C(A,W) = \textstyle\sum_{r=1}^{k} \left(\sum_{i\in A_r, j\in V\setminus A_r} W_{ij}\right) / \left(\sum_{i\in A_r, j\in V} W_{ij}\right).$$

We'll now re-write this problem a bit differently, to make the tie-in with the above algorithm clearer. Let $e_r \in \{0,1\}^n$ be the indicator vector for A_r; that is, $e_r^{(i)} = 1$ if the i'th data point is in the A_r'th cluster, and 0 otherwise. We have that:

$$\begin{aligned} e_r^T D e_r &= \textstyle\sum_{i\in A_r, j\in V} W_{ij} \\ e_r^T W e_r &= \textstyle\sum_{i\in A_r, j\in A_r} W_{ij} \\ e_r^T (D-W) e_r &= \textstyle\sum_{i\in A_r, j\in V\setminus A_r} W_{ij} \end{aligned}$$

Our goal is now to minimize:

$$C(e,W) = \textstyle\sum_{r=1}^k e_r^\top (D-W) e_r / (e_r^\top D e_r).$$

Notice that the 'normalizing factor', the denominator, biases towards big clusters. Let $E = (e_1, e_2, \ldots, e_r)$ be the matrix of cluster indicator vectors. A data point may only be assigned to one cluster, so we have the additional constraint that $E\vec{1} = \vec{1}$. Minimizing $C(e,W)$ with respect to these constraints is NP-hard. Next we'll present an equivalent formulation to our problem, but will introduce a relaxation that roughly corresponds to erasing the $\{0,1\}$ constraint. In this relaxed problem we'll be optimizing over a larger space, so we will find a better minimum. To get back to our original problem, we will have to do "rounding" (discussed in the next lecture).

Proposition 1 *$C(e,W)$ is equal to $k - \operatorname{tr} Y^\top D^{-1/2} W D^{-1/2} Y$, for any matrix Y such that (a) the columns of $D^{-1/2}Y$ are piecewise constant with respect to the clusters $\{A_r\}$, and (b) Y has orthonormal columns ($Y^\top Y = I$).*

Proof:

1. (a) $\Longrightarrow D^{-1/2}Y = (e_1, \ldots, e_R)\Lambda = E\Lambda$ for any Λ.
2. Re-write (b) as $I = Y^\top Y = \Lambda^\top (E^\top DE)\Lambda = \Lambda^\top (E^\top DE)^{1/2}(E^\top DE)^{1/2}\Lambda$. Note that D is diagonal and E is orthogonal, so $E^\top DE$ is diagonal. It is also strictly positive. Thus $E^\top DE$ has an inverse and a square root.
3. If you let $M = (E^\top DE)^{1/2}\Lambda$, note that the above is simply $I = M^\top M$. Thus, for square matrices M, we also have that $I = MM^\top$.
4. Substituting back M, $I = (E^\top DE)^{1/2}\Lambda\Lambda^T(E^\top DE)^{1/2} \Longrightarrow \Lambda\Lambda^T = (E^\top DE)^{-1}$.
5. $\operatorname{tr} Y^\top D^{-1/2}WD^{-1/2}Y = \operatorname{tr}\Lambda^\top E^\top WE\Lambda$. Permuting within a trace doesn't change its value, so this is equal to $\operatorname{tr} E^\top WE\Lambda\Lambda^\top$. Finally, substituting step 4's result, we get $\operatorname{tr}(E^\top WE)(E^\top DE)^{-1}$, which is equivalent to $\sum_{r=1}^k e_r^\top W e_r / (e_r^\top D e_r)$.

Now we relax the problem by dropping constraint (a) in the above formulation. Recall Raleigh's principle, e.g. $\max_x (x^T Ax)$ s.t. $x^T x = 1$ is A's largest eigenvector. Ky Fan's Theorem generalizes this for multiple vectors. The solutions to our relaxed problem,

$$\begin{gathered} \max \operatorname{tr} Y^\top D^{-1/2} W D^{-1/2} Y \\ \text{s.t. } Y^\top Y = I \end{gathered}$$

are all matrices $Y = UR$, where U is the matrix of the k largest eigenvectors of $D^{-1/2}WD^{-1/2}$, and R is any rotation matrix. Look back to the algorithm presented on page 1; it finds precisely this! The normalization and k-means performed in steps 2 & 3 do the rounding. Although the normalized cut framework provides some justification for using the algorithm originally presented, keep in mind that the algorithm is an *approximation*. For very small experiments, the normalized cut can be calculated exactly. Surprisingly, using the relaxed solution performs *better* than using the exact solution. No one really knows why.