Lecture topics:

- Spectral clustering, random walks and Markov chains

Spectral clustering

Spectral clustering refers to a class of clustering methods that approximate the problem of partitioning nodes in a weighted graph as eigenvalue problems. The weighted graph represents a similarity matrix between the objects associated with the nodes in the graph. A large positive weight connecting any two nodes (high similarity) biases the clustering algorithm to place the nodes in the same cluster. The graph representation is relational in the sense that it only holds information about the comparison of objects associated with the nodes.

Graph construction

A relational representation can be advantageous even in cases where a vector space representation is readily available. Consider, for example, the set of points in Figure 1a. There appears to be two clusters but neither cluster is well-captured by a small number of spherical Gaussians. By connecting each point to their two nearest neighbors (two closest points) yields a graph in Figure 1b that places the two clusters in different connected components. While typically the weighted graph representation would have edges spanning across the clusters, the example nevertheless highlights the fact that the relational representation can potentially be used to identify clusters whose form would make them otherwise difficult to find. This is particularly the case when the points lie on a lower dimensional surface (manifold). The weighted graph representation can essentially perform the clustering along the surface rather than in the enclosing space.

How exactly do we construct the weighted graph? The problem is analogous to the choice of distance function for hierarchical clustering and there are many possible ways to do this. A typical way, alluded to above, starts with a $k-$nearest neighbor graph, i.e., we construct an undirected graph over the n points such that i and j are connected if either i is among the k nearest neighbors of j or vice versa (nearest neighbor relations are not symmetric). Given the graph, we can then set

$$W_{ij} = \begin{cases} \exp(-\beta\|\mathbf{x}_i - \mathbf{x}_j\|) & \text{if } i \text{ and } j \text{ connected} \\ 0, \quad \text{otherwise} \end{cases} \tag{1}$$

The resulting weights (similarities) are symmetric in the sense that $W_{ij} = W_{ji}$. All the

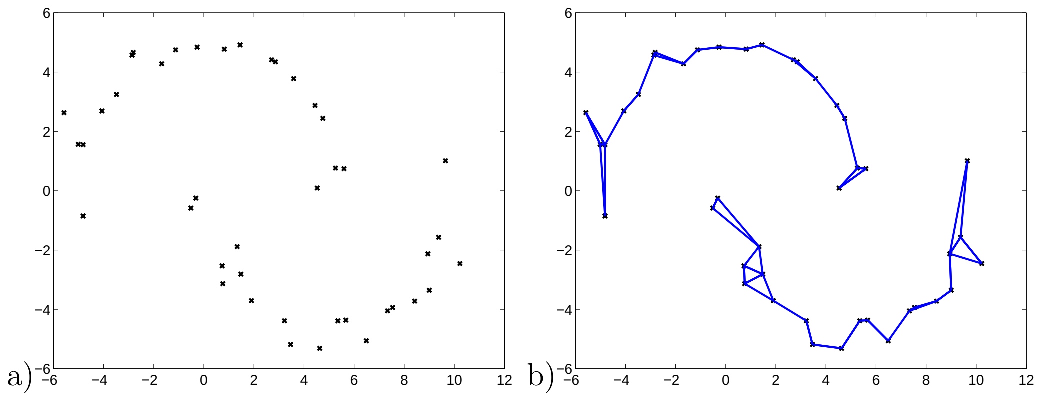


Figure 1: a) a set of points and b) the corresponding 2-nearest neighbor graph.

diagonal entries are set to zero: $W_{ii} = 0$ for $i = 1, \ldots, n$. The $n \times n$ matrix W now represents the weighted graph.

There are two parameters to set: k and β. The choice of k is tied to the dimensionality of the clusters we are trying to find. For example, if we believe that the clusters look like $d-$dimensional surfaces, then k should be at least d. A small k leads to a sparse graph and serves to limit the comparisons between points to those that are close. This is advantageous since the Euclidean distance is unlikely to be reasonable for points far away. For example, consider points on the surface of a unit sphere, and a context where their distance really should be measured along the surface. The simple Euclidean distance nevertheless provides a reasonable approximation for points that are close on the surface. β serves a similar role but, unlike k, is tied to the actual scale of the points (their distances).

Graph partitioning and criteria

Let's now define the clustering problem more formally. Suppose we have n objects to be clustered into two groups (binary partition). A multi-way partition can be obtained through a recursive application of binary partitioning. The objects are represented by a weighted graph with symmetric positive weights $W_{ij} = W_{ji} \geq 0$, W_{ij} is zero when no edge is present between i and j, and $W_{ii} = 0$. The goal is to use the weighted graph as a similarity measure to partition the nodes into two disjoint groups C^+ and C^- such that $C^+ \cup C^- = \{1, \ldots, n\}$. Any such partition corresponds to a labeling of the nodes in the graph with binary labels $y_i \in \{-1, 1\}$ such that, e.g., $y_i = 1$ when $i \in C^+$. The clusters can be therefore equivalently specified by the sets (C^+, C^-) or the labeling y (a binary vector of length n).

It remains to specify an objective function for finding the clusters. Each binary clustering is associated with a *cut* in the graph. The weight of the cut is given by

$$s(C^+, C^-) = \sum_{i \in C^+, j \in C^-} W_{ij} = \frac{1}{4} \sum_{i,j} W_{ij} (y_i - y_j)^2 = J(y) \tag{2}$$

where $(y_i - y_j)^2 = 4$ when y_i and y_j differ and zero otherwise. The cut simply corresponds to adding the weights of the edges connecting nodes that are in different clusters (labeled differently). The value of the cut is obviously zero if all the nodes are labeled the same. If we require both labels to be present we arrive at a *minimum cut* criterion. It is actually efficient to find the labeling or, equivalently, sets C^+ and C^- that minimize the value of the cut under this constraint. The approach does not work well as a clustering algorithm, however, as it tends to simply identify outliers as clusters (individual points weakly connected to others). We will have to modify the objective to find more balanced clusters.

A better criterion is given by so called *normalized cut* (see Shi and Malik 2000):

$$\text{Norm-cut}(C^+, C^-) = \frac{s(C^+, C^-)}{s(C^+, C^+) + s(C^+, C^-)} + \frac{s(C^+, C^-)}{s(C^-, C^-) + s(C^+, C^-)} \tag{3}$$

where, e.g., $s(C^+, C^+) = \sum_{i \in C^+, j \in C^+} W_{ij}$, the sum of weights between nodes in cluster C^+. Each term in the criterion is a ratio of the weight of the cut to the total weight associated with the nodes in the cluster. In other words, it is the fraction of weight tied to the cut. This normalization clearly prohibits us from separating outliers from other nodes. For example, an outlier connected to only one other node with a small weight cannot form a single cluster as the fraction of weight associated with the cut would be 1, the highest possible value of the ratio. So we can expect the criterion to yield more balanced partitions. Unfortunately, we can no longer find the solution efficiently (it is an integer programming problem). An approximate solution can be found by relaxing the optimization problem into an eigenvalue problem.

Spectral clustering, the eigenvalue problem

We begin by extending the "labeling" over the reals $z_i \in \mathcal{R}$. We will still interpret the sign of the real number z_i as the cluster label. This is a relaxation of the binary labeling problem but one that we need in order to arrive at an eigenvalue problem. First, let's rewrite the cut as

$$J(z) = \frac{1}{4}\sum_{i,j} W_{ij}(z_i - z_j)^2 = \frac{1}{4}\sum_{i,j} W_{ij}(z_i^2 - 2z_iz_j + z_j^2) = \frac{1}{4}\sum_{i,j} W_{ij}(2z_i^2 - 2z_iz_j) \quad (4)$$

$$= \frac{1}{2}\sum_i \overbrace{\left(\sum_j W_{ij}\right)}^{D_{ii}} z_i^2 + \frac{1}{2}\sum_{i,j} W_{ij}z_iz_j = \frac{1}{2}\, z^T(D-W)z \quad (5)$$

where we have used the symmetry of the weights. D is a diagonal matrix with elements $D_{ii} = \sum_j W_{ij}$. The matrix $L = D - W$ is known as the *graph Laplacian* and is guaranteed to be positive semi-definite (all the eigenvalues are non-negative). The smallest eigenvalue of the Laplacian is always exactly zero and corresponds to a constant eigenvector $z = \mathbf{1}$.

We will also have to take into account the normalization terms in the normalized cut objective. A complete derivation is a bit lengthy (described in the Shi and Malik paper available on the website). So we will just motivate here how to get to the relaxed version of the problem. Now, the normalized cut objective tries to balance the overall weight associated with the nodes in the two clusters, i.e., $s(C^+, C^+) + s(C^+, C^-) \approx s(C^-, C^-) + s(C^+, C^-)$. In terms of the labels, the condition for exactly balancing the weights would be $y^T D\mathbf{1} = 0$. We will instead use a relaxed criterion $z^T D\mathbf{1} = 0$. Moreover, now that z_i's are real numbers and not binary labels, we will have to normalize them so as to avoid $z_i \to 0$. We can do this by requiring that $z^T Dz = 1$. As a result, small changes in z_i for nodes that are strongly connected to others ($D_{ii} = \sum_j W_{ij}$ is large) require larger compensating changes at nodes that are only weakly coupled to others. This helps ensure that isolated nodes become “followers”.

The resulting relaxed optimization problem is given by

$$\text{minimize} \quad \frac{1}{2} z^T(D-W)z \quad \text{subject to} \quad z^T Dz = 1,\ z^T D\mathbf{1} = 0 \quad (6)$$

The solution can be found easily via Lagrange multipliers and reduces to finding the eigenvector z_2 (components z_{2i}) corresponding to the second smallest eigenvalue from

$$(D-W)z = \lambda Dz \ \text{ or, equivalently, } \ (I - D^{-1}W)z = \lambda z \quad (7)$$

The eigenvector with the smallest ($\lambda = 0$) eigenvalue is always the constant vector $z = \mathbf{1}$. This would not satisfy $z^T D\mathbf{1} = 0$ but the second smallest eigenvector does. Note that since the goal is to minimize $z^T(D-W)z$ we are interested in only the eigenvectors with small eigenvalues. The clusters can now be found by labeling the nodes according to

$\hat{y}_i = \text{sign}(z_{2i})$. If we wish to further balance the number of nodes in each cluster, we could sort the components of z_2 in the ascending order and label nodes as negative in this order. Figure 2 illustrates a possible solution and the corresponding values of the eigenvector.

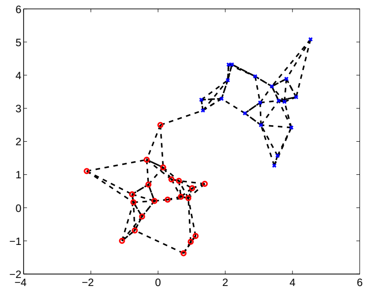
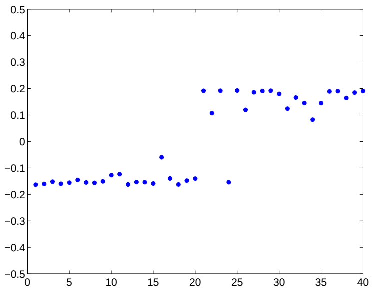
a)  b) 

Figure 2: a) spectral clustering solution and b) the values of the second largest eigenvector.

Spectral clustering, random walk

The relaxed optimization problem is an approximate solution to the normalized cut problem. It is therefore not immediately clear that this approximate solution behaves appropriately. We can try to justify it from a very different perspective, that of random walks on the weighted graph. To this end, note that the eigenvectors we get by solving $(I - D^{-1}W)z = \lambda z$ are exactly the same as those obtained from $D^{-1}Wz = \lambda' z$. The resulting eigenvalues are also in one to one correspondence: $\lambda' = 1 - \lambda$. Thus the constant eigenvector $z = \mathbf{1}$ with $\lambda = 0$ should have $\lambda' = 1$ and satisfy $D^{-1}Wz = z$. Let's understand this further. Define

$$P_{ij} = \frac{W_{ij}}{\sum_{j'} W_{ij'}} = \frac{W_{ij}}{D_{ii}} \tag{8}$$

so that $P = D^{-1}W$. Clearly, $\sum_j P_{ij} = 1$ for all i so that $P\mathbf{1} = \mathbf{1}$. We can therefore interpret P as a *transition probability matrix* associated with the nodes in the weighted graph. In other words, P_{ij} defines a random walk where we hop from node i to node j with probability P_{ij}. If $X(t)$ denotes the node we happen to be at time t, then

$$P(X(t+1) = j | X(t) = i) = P_{ij} \tag{9}$$

Our random walk corresponds to a *homogeneous Markov chain* since the transition probabilities remain the same every time we come back to a node (i.e., the transition probabilities are not time dependent). Markov chains are typically defined in terms of states and transitions between them. The states in our case are the nodes in the graph.

In a Markov chain two states i and j are said to be *communicating* if you can get from i to j and from j to i with finite probability. If all the pairs of states (nodes) are communicating, then the Markov chain is *irreducible*. Note that a random walk defined on the basis of the graph in Figure 1b would not be irreducible since the nodes across the two connected components are not communicating.

It is often useful to write a *transition diagram* that specifies all the permissible one-step transitions $i \to j$, those corresponding to $P_{ij} > 0$. This is usually a directed graph. However, in our case, because the weights are symmetric, if you can directly transition from i to j then you can also go directly from j to i. The transition diagram therefore reduces to the undirected graph (or directed graph where each undirected edge is directed both ways). Note that the transition probabilities themselves are not symmetric as the normalization terms D_{ii} vary from node to node. On the other hand, the zeros (prohibited one-step transitions) do appear in symmetric places in the matrix P_{ij}.

We need to understand one additional property of (some) Markov chains – *ergodicity*. To this end, let us consider one-step, two-step, and $m-$step transition probabilities:

$$
\begin{aligned}
P(X(t+1) = j|X(t) = i) &= P_{ij} && (10)\\
P(X(t+2) = j|X(t) = i) &= \sum_k P_{ik}P_{kj} = [PP]_{ij} = [P^2]_{ij} && (11)\\
&\cdots && (12)\\
P(X(t+m) = j|X(t) = i) &= [P^m]_{ij} && (13)
\end{aligned}
$$

where $[P^m]_{ij}$ is the i, j element of the matrix $PP\cdots P$ (m multiplications). A Markov chain is *ergodic* if there is a finite m such that for this m (and all larger values of m)

$$P(X(t+m) = j|X(t) = i) > 0, \text{ for all } i \text{ and } j \tag{14}$$

In other words, we have to be able to get form any state to any other state with finite probability after m transitions. Note that this has to hold for the same m. For example, a Markov chain with three states and possible transitions $1 \to 2 \to 3 \to 1$ is not ergodic even though we can get from any state to any other state. However, for this Markov chain, any m step transition probability matrix would still have prohibited transitions. For example, starting from 1, after three steps we can only be back in 1.

Now, what will happen if we let $m \to \infty$, i.e., follow the random walk for a long time? If the Markov chain is ergodic then

$$\lim_{m\to\infty} P(X(t+m) = j|X(t) = i) = \pi_j \tag{15}$$

for some *stationary distribution* π. Note that π_j does not depend on i at all. In other words, the random walk will forget where it started from. Ergodic Markov chains ensure that there's enough "mixing" so that the information about the initial state is lost. In our case, roughly speaking, any connected graph gives rise to an ergodic Markov chain.

Back to clustering. The fact that a random walk on the graph forgets where it started from is very useful to us in terms of identifying clusters. Consider, for example, two tightly connected clusters that are only weakly coupled across. The random walk started at a node in one of the clusters quickly forgets which state within the cluster it begun. However, the information about which cluster the starting node was in lingers much longer. It is precisely this lingering information about clusters in random walks that helps us identify them. This is also something we can understand based on eigenvalues and eigenvectors.

So, let's try to identify clusters by seeing what information we have about the random walk after a large number of steps. To make our analysis a bit easier, we will rewrite

$$P^m = D^{-1/2}(D^{-1/2}WD^{-1/2})^m D^{1/2} \tag{16}$$

You can easily verify this for $m = 1, 2$. The symmetric matrix $D^{-1/2}WD^{-1/2}$ can be written in terms of its eigenvalues $\lambda'_1 \geq \lambda'_2 \geq \ldots$ and eigenvectors $\tilde{z}_1, \tilde{z}_2, \ldots$

$$(D^{-1/2}WD^{-1/2})^m = (\lambda'_1)^m \tilde{z}_1\tilde{z}_1^T + (\lambda'_2)^m \tilde{z}_2\tilde{z}_2^T + \ldots + (\lambda'_n)^m \tilde{z}_n\tilde{z}_n^T \tag{17}$$

The eigenvalues are the same as those of P and any eigenvector $\tilde{z}$ of $D^{-1/2}WD^{-1/2}$ corresponds to an eigenvector $z_2 = D^{-1/2}\tilde{z}_2$ of P. As $m \to \infty$, clearly

$$P^\infty = D^{-1/2}\tilde{z}_1\tilde{z}_1^T D^{1/2} \tag{18}$$

since $\lambda'_1 = 1$ as before and $\lambda'_2 < 1$ (ergodicity). The goal is to understand which transitions remain strong even for large m. These should be transitions within clusters. So, since the eigenvalues are ordered, for large m

$$P^m \approx D^{-1/2}\left(\tilde{z}_1\tilde{z}_1^T + (\lambda'_2)^m \tilde{z}_2\tilde{z}_2^T\right) D^{1/2} \tag{19}$$

where $\tilde{z}_2$ is the eigenvector with the second largest eigenvalue. Note that its components have the same signs as the components of z_2, the second largest eigenvector of P. Let's look at the "correction term" $(\tilde{z}_2\tilde{z}_2^T)_{ij} = \tilde{z}_{2i}\tilde{z}_{2j}$. In other words, we get lingering stronger transitions between i and j corresponding to nodes where $\tilde{z}_{2i}$ and $\tilde{z}_{2j}$ have the same sign, and decreased transition across. These are the clusters and indeed obtained by reading the cluster assignments from the signs of the components of the relevant eigenvector.