CH4: Clustering Theory and Spectral Clustering

CS 182 Spring 2026

Scribes: chuang25, xqian6, spurkaya, jyuan17, dbenisvy, cmeyer5, mgoetz2

Compiled & edited by eyouth, cbaker20, dhan30

Reach out to cs1820tas@lists.brown.edu for any clarifications or corrections.

Overview of Clustering

Clustering is a rapidly-developing area of computational biology that enables unsupervised inference of complex patterns from large data sets. It has many useful applications from gene expression analysis to population genetic inference to document categorization, and diverse methods have been developed to extract meaning from large-scale bioinformatic data.

Introduction to Clustering Theory

Definition of a Clustering Problem

A clustering problem involves the following elements:

- a *space* of elements (may be Euclidean or non-Euclidean)
- an *input data set* of *points* belonging to the space
- a *distance metric* which can be applied to any pair of points within the space

Definition of a Distance Metric

A *distance metric* is a function $d(x, y)$ which takes two points in the clustering space as inputs, produces a real number as output, and satisfies the following axioms:

Axiom 1: Nonnegativity of distances: $d(x, y) \geq 0$ for all x, y

Axiom 2: Property of zero distance: $d(x, y) = 0$ if and only if $x = y$

Axiom 3: Symmetry of distances: $d(x, y) = d(y, x)$ for all x, y

Axiom 4: Triangle inequality: $d(x, y) \leq d(x, z) + d(y, z)$ for all x, y, z

Low-Dimensional Euclidean Spaces

In low-dimensional Euclidean spaces, points are vectors of real numbers (*coordinates*). The number of coordinates is the dimension of the space. For example, $\mathbb{R}^n$ is the n-dimensional real numbers vector space.

Note that the average of a set of points in Euclidean space always exists as a point within the space.

Distance Metrics in Low-Dimensional Euclidean Space

- L_r-norm:

$$d([x_1, \ldots, x_n], [y_1, \ldots, y_n]) = \left(\sum_{i=1}^{n} |x_i - y_i|^r \right)^{\frac{1}{r}}$$

  Various common distance metrics can be derived from the general L_r-norm.

- *Euclidean distance (L_2-norm)*: the square root of the sum of the squared differences between the coordinates of two points in each dimension:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- *Manhattan distance (L_1-norm)*: the sum of the absolute magnitudes of the differences between the coordinates of two points in each dimension:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- *Chebyshev distance (L_∞-norm)*: the maximum absolute magnitude of the difference between the coordinates of two points in any dimension:

$$d(x, y) = \underset{i \in \{1, \ldots, n\}}{\text{MAX}} |x_i - y_i|$$

- *Pearson correlation coefficient*: the signed correlation between the coordinates of two points in each dimension (note that this metric would need to be rescaled to satisfy the first axiom above)

Non-Euclidean Spaces

High-dimensional Euclidean spaces are not really Euclidean spaces at all. Example high-dimensional clustering problems include document clustering by topic based on occurrence of common or unusual words, social media community clustering by specific preferences (e.g., types of movies that movie-goers like), and biological sequence clustering based on similarity/distance metrics computed between strings.

Note that it does not make sense to define an "average" point in the context of sets or strings, as it does for Euclidean spaces.

Distance Metrics in Non-Euclidean Spaces

- *Jaccard distance*: the complement of the ratio of the sizes of the intersection and the union of two sets:

$$d(x, y) = 1 - \frac{|x \cap y|}{|x \cup y|}$$

  The ratio above is also known as the *Jaccard similarity* between two sets.

- *cosine distance*: the angle between two vectors in a dimensional space (may be Euclidean or not):

$$d(x, y) = \text{ARCCOS} \frac{|x \cdot y|}{||x|| \cdot ||y||}$$

- *Hamming distance*: the number of coordinates at which two vectors differ (often used in Boolean or other discrete spaces)
- *edit distance*: the smallest number of single-character insertions and deletions which will convert one string into another (within the space of strings)

The Curse of Dimensionality

In high-dimensional Euclidean spaces:

- almost all pairs of points are equally far away from one another
- almost any two vectors are almost orthogonal

This necessitates appropriate definition and application of distance metrics to extract meaningful insights about the relationships between points in the space.

Hierarchical Clustering

Hierarchical clustering algorithms are simple agglomerative methods of clustering points into closer and farther groups. In Euclidean spaces, such algorithms employ the notion of a *centroid* and are best applied to relatively small data sets. In non-Euclidean spaces, such algorithms employ the notion of a *clustroid* and the cluster representation must be tailored to the space of interest.

Intuitively, hierarchical clustering algorithms initialize the input data as single-point clusters, and progress by successively combining nearby clusters according to some distance metric.

Definition of a Hierarchical Clustering Problem

A hierarchical clustering problem involves the following elements:

- *cluster representation*: how are clusters represented in the space of interest?
- *merging rule*: how to determine which clusters to merge at a given step?
- *stopping rule*: how to decide when the algorithm should stop merging clusters?

Hierarchical Clustering in Euclidean Spaces

Cluster Representation

In a Euclidean space, a cluster can be represented by its *centroid*, or the average of all the points in the cluster. The centroid of a cluster containing a single point is therefore just that point. The distance between clusters can then be computed as the distance between their centroids.

Merging Rule

There are several methods of selecting the clusters to merge at each step of the algorithm. For example, the two clusters whose centroids are closest together (based on some distance metric) may be merged at each step, with ties broken arbitrarily. The centroid of the resulting cluster may then be computed as the average of all the points from both clusters merged.

Alternative rules include merging the two clusters who exhibit the minimum distance between any two points, the minimum average distance between all pairs of points from each cluster, the minimum resulting *radius* (maximum distance between the new centroid and any point within the merged cluster), or the minimum resulting *diameter* (maximum distance between any two points in the new cluster). Variations on radius- or diameter-based distance metrics may include average or sum-of-squares computations.

Stopping Rule

Clustering may be terminated when the total number of distinct clusters has been reduced to k (for some predetermined natural number k). Commonly, $k = 1$ is used, to ensure that all clusters are merged hierarchically. Alternatively, the algorithm may be terminated when the next merger proposed is determined to be "inadequate"; e.g., if cluster radius or diameter exceeds some threshold, or if cluster *density* (the ratio of the number of points in the cluster divided by the radius or diameter raised to some power) falls below some threshold. Tracking the average diameter or density across merge steps can also identify the optimal number of clusters prior to a "jump" in one of these metrics, which would signify a "bad" cluster.

Interpretation of Results

The output of a hierarchical clustering algorithm is a tree which represents the order in which all points were combined through successive mergers. This representation is inspired by phylogeny tree construction algorithms in which mergers represent evolutionary branching events and distance measures approximate evolutionary time.

Algorithmic Complexity

Hierarchical clustering algorithms are not very efficient, as they must compute the distances between each pair of clusters at every step in order to determine which pair to merge. As these computations are quadratic in the number of clusters at each step, the total runtime of the algorithm is $O(n^3)$. This cubic algorithm is computationally intensive and therefore can only be applied to a relatively small number of points.

However, the overall runtime of hierarchical clustering algorithms can be improved by use of appropriate data structures. By placing all pairs of points and their corresponding distances into a priority queue, the smallest distance can always be found in a single step. Dynamically deleting and inserting new entries into this queue as necessary following each merger can reduce the overall runtime to $O(n^2 \log n)$, which is faster than the naive algorithm (although still fairly computationally intensive for large values of n).

Hierarchical Clustering in Non-Euclidean Spaces

Cluster Representation

In a non-Euclidean space, the centroid may not be directly representable as a point in the space, necessitating the use of *clustroids*. The clustroid of a cluster is simply one of its points, typically chosen so as to minimize its distance to all other points in the cluster (based on some distance metric). The algorithmic design described above may then be adjusted to incorporate this representation.

Merging Rule

The same methods employed in Euclidean spaces may be used to select pairs of clusters to merge at each step, with the replacement of centroids by clustroids. The minimum or average distance between pairs of points in different clusters may still be used as in Euclidean spaces. The radius of a cluster may be defined in terms of its clustroid, while the diameter may be computed as in Euclidean spaces.

Stopping Rule

Clustering may be terminated using the same methods as in Euclidean spaces.

Challenges of Clustering in High-Dimensional Spaces

In high-dimensional spaces, almost all points will have distances close to the average distance, which complicates the selection of "closest" clusters at each step. While there may be high-dimensional structure within real data sets, it can still be quite difficult to extract meaningful clusters from pairs of points that are approximately the same distance apart in the space. More sophisticated clustering algorithms are therefore required to effectively and meaningfully cluster high-dimensional data.

k-Means Clustering

The k-means clustering algorithm is the best-known member of the family of "point assignment clustering" algorithms. Like hierarchical clustering and other "tree construction"/"flat" algorithms, these algorithms produce *hard clusterings* (i.e., all points are unambiguously assigned to individual clusters).

Assumptions

1. The clustering space is Euclidean
2. The number of clusters (k) is provided in advance

Pseudocode

Algorithm 1 k-Means Clustering Algorithm

```
procedure KMEANS(x = (x_1, x_2, ...x_n), k)
    Initialize centroids for k clusters
    while still converging do
        Assign all points to the cluster with the closest centroid
        Recalculate all cluster centroids based on reassignment
    return k clusters and their corresponding points
```

Intuitively, the algorithm successively "tunes" the centroids of each cluster and the cluster membership of each point through iterative reassignment of points and recalculation of cluster centroids, ultimately converging to a stable cluster structure.

Proof of Convergence

The *residual sum of squares* (RSS) is the squared distance of each vector from its cluster's centroid:

$$\mathsf{RSS} = \sum_{i=1}^{k} \left[\sum_{x \in C_i} (x - \mu_i)^2 \right]$$

where $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$ is the centroid of cluster i (C_i). RSS is therefore the objective function which the k-means clustering algorithm seeks to minimize, as smaller values of RSS indicate greater quality of clustering assignments (i.e., smaller average squared distances between all points and their corresponding cluster centroids). RSS decreases in both the point reassignment and cluster centroid recalculation steps, because each point is successively assigned to the centroid which minimizes its contribution to the overall RSS, and the new centroid of each cluster is calculated to minimize the cluster-wise RRS (to see this, differentiate the inner summand above with respect to μ_i and observe that the formula which results is exactly the definition of the centroid given above).

Because there are only a finite number of possible clusterings, a monotonically-decreasing algorithm will eventually arrive at a local minimum (provided that any ties are broken so as to avoid infinite cycles). The k-means clustering algorithm is not guaranteed to find the global minimum of the RSS; however, a local minimum can be easily determined by instituting some convergence criterion based on the degree to which cluster centroids change each iteration (i.e., the algorithm may be terminated once the absolute change in cluster centroids from one iteration to the next falls below some predetermined threshold).

Algorithmic Complexity

Computation of the distance between each point-centroid pair, reassignment of points, and recalculation of cluster centroids can be carried out in linear time with respect to the number of points n (if the number of

dimensions and clusters are small constants). If the number of iterations until convergence is also fixed or bounded by some constant, the overall runtime of the k-means clustering algorithm is $O(n)$.

Initialization

There are multiple methods of selecting an initial k "seeds" to initialize the algorithm. The points may first be clustered hierarchically into k clusters, and the point closest to each centroid may be selected from each cluster to yield k points that are relatively spread out. Alternatively, the first point may be selected randomly and for $i = 2, \ldots, k$ the point which is the maximal minimum distance from all existing "seeds" may be selected from the remaining set of points.

Outliers present problems for k-means clustering initialization, as selection of an outlier point as a "seed" may result in the inference of a singleton cluster which contains only that point. Avoiding outlier points in the "seeding" phase is therefore important.

k-Optimization Methods

There are also multiple methods of tuning the value of k to produce meaningful clusters. The concept of *diffuseness* (as quantified by average radius or diameter across all clusters) may be used to identify an optimal number of clusters which does not cause either metric to "jump" (analogous to detection of "inadequate" or "bad" clusters in determining when to terminate hierarchical clustering algorithms). Alternatively, a binary search for the "best" value of k can select an optimal number of clusters in logarithmic time.

EM Clustering

EM clustering methods make use of the Expectation-Maximization algorithm to iteratively refine clusterings. Unlike hierarchical clustering and k-means clustering (which both produce "hard" clusterings), EM clustering algorithms produce "soft" clusterings (i.e., each point may be assigned to multiple clusters with varying degrees of likelihood, forming a probability distribution over the cluster space). The "E-step" and "M-step" depend upon the design of the specific clustering problem.

Examples of EM clustering models include *Bernoulli models* and *mixture models*. Such models are useful in clustering documents based on word content, among other applications.

Spectral Graph Theory

Many clustering problems (e.g., social network modeling, gene expression analysis, etc.) involve high-dimensional data which is typically represented as very large matrices. It is therefore beneficial to "condense" such data into a smaller number of dimensions for greater ease and accuracy of analysis.

Linear Algebra Review

Properties of a matrix M with eigenvalues $\lambda(M)$

Defn: A square matrix M has *eigenvalue* ($\lambda \in \mathbb{C}$) and corresponding *eigenvector* (e) if $Me = \lambda e$.

For example, if $M = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$, one eigenvalue is $\lambda = 7$ with corresponding (unit) eigenvector $e = \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{bmatrix}$.

The m *eigenpairs* of $M_{m \times m}$ can be computed in cubic time ($O(m^3)$).

Defn: A matrix M is symmetric if $M^T = M$.

Lemma: If $M^T = M$, then $\lambda(M)$ are real numbers $\lambda(M) = \{\lambda_i(M) | 1 \leq i \leq n\}$.

Defn: A matrix M is positive definite ($M \succ 0$) if $x^T M x > 0, \forall x \neq \vec{0}$ and positive semidefinite ($M \succeq 0$) if $x^T M x \geq 0, \forall x \neq \vec{0}$.

Lemma: A matrix M is positive definite if and only if $\lambda_i(M) > 0, \forall i$.

Proof (forward direction): Suppose $\lambda_j = 0$. Then, λ_j has associated eigenvector v_j such that $v_j^T M v_j = v_j^T((0)v_j) = \vec{0}$. By contraposition, we have shown that if M is positive definite, $\lambda_i(M) > 0, \forall i$.

Lemma: For a given matrix B, $M = B^T B$ is positive semidefinite.

Proof: $x^T M x = x^T B^T B x = (Bx)^T Bx = \|Bx\|_F^2 \geq 0$. Note that this final representation is the Frobenius norm, i.e. $\langle Bx, Bx \rangle$ (dot product with itself).

Eigenpair Computation Using the Characteristic Polynomial

The eigenvalues of a matrix $M_{m \times m}$ can be computed by observing that

$$Me = \lambda e$$

$$\implies Me - \lambda e = 0$$

$$\implies (M - \lambda I)e = 0$$

$$\implies \det(M - \lambda I) = 0$$

where I is the $m \times m$ identity matrix.

The determinant of $M - \lambda I$ is an n^{th}-degree polynomial, known as the *characteristic polynomial*. Solving for the zeros of this polynomials yields the m eigenvalues of M. For the example above:

$$M - \lambda I = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\implies \det(M - \lambda I) = \begin{vmatrix} 3 - \lambda & 2 \\ 2 & 6 - \lambda \end{vmatrix} = 0$$

$$\implies (3 - \lambda)(6 - \lambda) - 4 = 0$$

$$\implies \lambda^2 - 9\lambda + 14 = 0$$

$$\implies (\lambda - 7)(\lambda - 2) = 0$$

$$\implies \lambda = 7 \text{ or } 2$$

The eigenvalues of M can then be computed by solving the system of linear equations resulting from substituting each λ_i into the equation $Me = \lambda e$ to obtain e_i (or its unit equivalent $\hat{e}_i$). For example:

$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} e_1 = 7e_1 \implies \begin{cases} 3e_{11} + 2e_{12} = 7e_{11} \\ 2e_{11} + 6e_{12} = 7e_{12} \end{cases}$$

$$\implies e_{12} = 2e_{11} \qquad \implies 2e_{22} = -e_{21}$$

$$\implies e_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \implies e_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

$$\implies \hat{e}_1 = \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{bmatrix} \qquad \implies \hat{e}_2 = \begin{bmatrix} \frac{2}{\sqrt{5}} \\ -\frac{1}{\sqrt{5}} \end{bmatrix}$$

$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} e_2 = 2e_2 \implies \begin{cases} 3e_{21} + 2e_{22} = 2e_{21} \\ 2e_{21} + 6e_{22} = 2e_{22} \end{cases}$$

Eigenpair Computation Using the Power Iteration Algorithm

The *power iteration algorithm* enables computation of the m eigenvalues of a matrix $M_{m \times m}$ in descending order (i.e., largest to smallest). The algorithm initializes a nonzero vector x_0 and iterates as follows:

$$x_{k+1} = \frac{Mx_k}{||Mx_k||}$$

x_k will eventually converge (i.e., $||x_{k+1} - x_k|| < \epsilon$ for some constant $\epsilon > 0$) to some vector e_1. e_1 is then approximately the principal eigenvector of M (i.e., an eigenvector corresponding to the largest eigenvalue λ_1). This eigenvalue can then be computed by manipulating the eigenpair equation as follows:

$$Me_1 = \lambda_1 e_1$$

$$\implies \lambda_1 = e_1^\top M e_1$$

Subsequent eigenpairs can be computed by transforming the problem so that the "contribution" of λ_1 is effectively negated and λ_2 becomes the "new" principal eigenvector:

$$M^* = M - \lambda_1 e_1 e_1^\top$$

Performing the algorithm on this matrix will then yield λ_2 and e_2. Similar iterations can be performed to compute the remaining eigenpairs.

Properties of Eigenvectors

Eigenvectors have unit norm (in standard form) and are mutually orthonormal:

$$||e_i|| = 1 \text{ for all } i \in \{1, \ldots, m\}$$

$$e_i \cdot e_j = 0 \text{ for all } i, j \in \{1, \ldots, m\}$$

The matrix of (unit) eigenvectors (E) satisfies the following property:

$$EE^\top = E^\top E = I_{m \times m}$$

Eigenvalue Decomposition

If we construct a matrix of eigenvectors $V = [v_1, v_2 ... v_n]$ for a matrix M such that $MV = V\Lambda$, where

$$\Lambda = \begin{bmatrix} \lambda_1 & ... & 0 \\ \vdots & \ddots & \vdots \\ 0 & ... & \lambda_n \end{bmatrix}$$

we can decompose M if and only if V is invertible, i.e. $M = V\Lambda V^{-1}$. V is invertible if and only if the eigenvectors are linearly independent. Note that V and Λ may both be complex in this setting; in other words, $M \in \mathbb{R}^{n \times n}$, but $V, \Lambda \in \mathbb{C}^{n \times n}$. We can examine how this works in the nice case that M is symmetric.

If M is symmetric, $\lambda_i \in \mathbb{R}, \forall i$ and v_i, v_j satisfy $\langle v_i, v_j \rangle = 0$ for all $i \neq j$ (all orthogonal and linearly independent). Let V be the matrix of unit eigenvectors in this setting. Then, $V^T V = I$, by the orthogonality of eigenvectors and that $v_i^T v_i = 1$ (Frobenius norm of a unit vector). In this case, note that $V^T = V^{-1}$.

Define the orthogonal group of dimension n as $\mathcal{O}(n) = \{R \in \mathbb{R}^{n \times n} | R^T R = I\}$. Then, if M is symmetric with dimension $n \times n$, $V \in \mathcal{O}(n)$, and we recover $M = V\Lambda V^T$.

This decomposition has shortcomings: it doesn't always exist, has $n \times n$ matrices (bad for memory and runtime), and may have complex values.

Principal Component Analysis

Principal component analysis (PCA) is a method by which a data set of points in high-dimensional space are "projected" into a (much) lower number of dimensions for analysis. These dimensions are those which explain the most variation within the original data set. Visualization or clustering of the same points based on only the first few "principal components" can enable more straightforward analysis relative to the original high-dimensional space.

The general procedure is outlined as follows:

1. Create a matrix M with each data point as a row
2. Compute the eigenvectors of the symmetric matrix $M^\top M$ or $MM^\top$
3. "Project" the data points into a lower-dimensional space whose axes are the first k eigenvectors

The principal eigenvector forms the axis along which the variance of the data is maximized. Subsequent eigenvectors form the axes along which the remaining variance of the data is maximized.

Dimensionality Reduction

Multiplying the original matrix of points M by the eigenvector matrix V rotationally transforms the data into a new coordinate system whose axes are the eigenvectors. If only the first k columns of V (eigenvectors of M) are used, the data can be dimensionally-reduced by considering only the k "components" which account for the greatest proportion of variance in the original data set.

Properties of PCA

Essentially equivalent results will be obtained by using either $M^\top M$ or $MM^\top$ to compute eigenvectors. For example, if M is a 4×2 matrix, the first two eigenvalues of $MM^\top$ (a 4×4 matrix) will be the same as the eigenvalues of $M^\top M$ (a 2×2 matrix), and the remaining two eigenvalues will be equal to 0. There will also be a relationship between the first two eigenvectors of each matrix.

Additionally, from the eigenpair equation ($M^\top M e = \lambda e$) it follows that

$$M^\top M V = VL$$

where L is a matrix with the eigenvalues of $M^\top M$ on the diagonal in descending order and 0s elsewhere.

Singular Value Decomposition

Singular value decomposition (SVD) is a method of matrix factorization which enables high-dimensional data to be represented in low-dimensional spaces. Given a matrix $M_{m\times n}$ with rank r, a decomposition of the following form can be found:

$$M = U\Sigma V^\top$$

where U is an $m \times r$ matrix, Σ is an $r \times r$ matrix, and V is an $n \times r$ matrix. These matrices reveal the underlying structure present in the data. The *latent size* r corresponds to labels or categories which characterize the variation within the data. For example, if M were a matrix of m individuals ranking n movies on a scale, U would contain eigenvectors relating individuals to movie concepts, Σ would contain eigenvalues corresponding to the importance of each of the r movie concepts in explaining the variation within the data, and V would contain eigenvectors relating movies to movie concepts.

Notably, small eigenvectors may be identified from the Σ matrix and discarded in order to perform dimensionality reduction.

To understand how this relates to eigenvalue decomposition, consider $M \in \mathbb{R}^{m\times n}$. There exist U, V, Σ such that $U \in \mathcal{O}, V \in \mathcal{O}$. Σ is diagonal $m \times n$ with diagonal $\sigma_1, \sigma_2, ...\sigma_r, 0, ...0$. Note $\sigma_1 \geq \sigma_2 \geq ...\sigma_r$, where $r = rank(M)$. The decomposition is $M = U\Sigma V$, where $U = [U_1, ...U_m]$ are left singular values of M, while $V = [V_1, ...V_n]$ are right singular values of M. We call $\sigma_1, ...\sigma_r$ the singular values of M.

In order to make this representation more compact, we can trim off the last $n - r$ rows of V and $m - r$ rows of U. In particular, $M = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ where u_i is a vector of length m and v_i is a vector of length n. Note that this trimming does not effect our product, as the rest of the latent multiplication would output all zeros.

Also, consider the following rearrangements of SVD in the case where M is symmetric:

$$\begin{aligned} MM^T &= U\Sigma V^T V\Sigma^T U^T \\ &= U\Sigma^2 U^T \end{aligned}$$

$$\begin{aligned} M^T M &= V\Sigma U^T U\Sigma V^T \\ &= V\Sigma^2 V^T \end{aligned}$$

So, the eigenvalues of $A^T A$ are the singular values of A squared.

Spectral Clustering

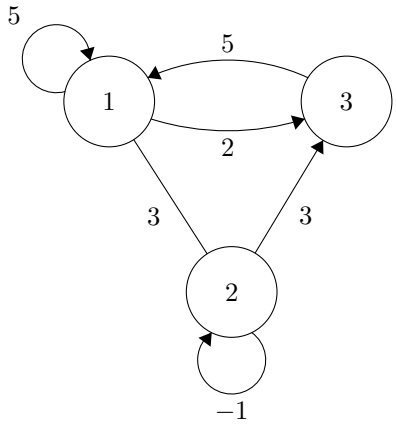
Given a data set of points and a nonnegative similarity function s_{ij}, an ideal clustering algorithm would be able to accurately divide all points into several groups such that both

- points in the same cluster are relatively similar to one another
- points in different clusters are relatively dissimilar to one another

A *weighted adjacency graph* $G(V, E)$ may be constructed for an input data set, such that each vertex v_i represents a data point and each edge $e_{ij} = (v_i, v_j)$ has weight s_{ij}. In such a graph, high-weight edges indicate high similarity between vertices, and low-weight (or zero-weight) edges indicate low (or no) similarity

between vertices. The weights of each edge in the graph can be represented by a square matrix $W = \{w_{ij}\}$, where each weight is given by the output of the similarity function applied to a pair of points. The choice of how to compute W is examined in further detail in von Luxburg's tutorial. See the following equivalent weighted adjacency matrix and graph for more clarity:

$$\begin{bmatrix} 5 & 3 & 2 \\ 3 & -1 & 3 \\ 5 & 0 & 0 \end{bmatrix}$$



The *degree matrix* D of W is designed to understand the total outflow/inflow each node i; in particular, we generally define $D = diag(\sum_{j=1}^{n} W_{ij})$. However, D can also be defined as $D = diag(\sum_{j=1}^{n} [W_{ij} > 0])$, in addition to other notions.

The unnormalized *Laplacian* of the graph G is defined as $D - W$, for our spectral clustering purposes. L has a number of nice properties that are explored more in von Luxburg's tutorial, but we'll formalize a few here. Consider the notion of an indicator vector $\mathbb{1}_A$. In particular, consider a set of vertices $A \subseteq V$. Then, $\mathbb{1}_A = (f_1, ...f_n) \in \mathbb{R}^n$, $f_i \in \{0, 1\}$, such that

$$f_i = \begin{cases} 1 \text{ if } v_i \in A \\ 0 \text{ otherwise} \end{cases}$$

Using this notation, unnormalized L has the following properties:

1. For every $f \; in \mathbb{R}^n$, $f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2$. This is equal to the mean cut magnitude in the graph.
2. L is symmetric and positive semi-definite.
3. The smallest eigenvalue of L is 0, and the corresponding eigenvector is all ones (for a fully connected graph).
4. L has n non-negative real valued eigenvectors.

We can additionally use the notion of a normalized Laplacian, which we can defined as $L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ or as $L_{rw} = D^{-1} L = I - D^{-1} W$. We can use the latter to think about random walks on W. In particular, the probability of moving from v_i to v_j is indicated by the result of computing $D^{-1} W$.

Spectral clustering is a method by which this concept of a weighted adjacency graph can be coupled with dimensionality reduction techniques to produce high-quality clusterings of high-dimensional data. It is named for its utilization of the *spectrum* (set of eigenvalues) of the similarity matrix of the data.

Pseudocode

Algorithm 2 Spectral Clustering Algorithm

procedure SPECTRAL($x = (x_1, x_2, ...x_n), k$)
- Construct a similarity matrix $S_{n\times n}$
- Construct a weighted adjacency matrix $W_{n\times n}$ by some modification of S
- Construct a diagonal degree matrix $D_{n\times n} = \sum_{j=1}^{n} W_{ij}$ (or according to some other heuristic)
- Compute the unnormalized Laplacian matrix $L_{n\times n} = D - W$
- Compute the first[1] k eigenvectors $(e_1, \ldots, e_k)$ of L
- Construct the matrix $U_{m\times k}$ whose columns are $e_1, \ldots, e_k$
- Cluster the n rows of U into k groups using the k-means algorithm (i.e., run KMEANS(U, k))
- **return** the k resulting row-wise clusters

Variations

The results of the spectral clustering algorithm will depend upon the specific similarity function s_{ij} employed. The adjacency matrix W may be constructed from the similarity matrix S by filtering based on some constant similarity threshold, only retaining edges between points which are (mutual) k-nearest neighbors, or weighting all edges by Gaussian similarity or some other metric. There are also several methods of normalizing the Laplacian L, each of which may alter the results obtained.

Applications

Spectral graph theory, spectral decomposition and spectral clustering algorithms all have applications in the areas of graph cuts, relaxation optimization problems, random walk theory, and perturbation theory. Given the high-dimensionality of many data sets within the field of computational biology, such methods which utilize linear algebra to perform dimensionality reduction and iterative clustering are of great interest to the future of bioinformatics and medical research, and can enable profound insights across a range of applications!

[1] Note that in this context the "first" k eigenvectors correspond to the k *smallest* (nonzero) values, as opposed to in PCA (for which the *largest* eigenvalues are most important).