CS 412 — Introduction to Machine Learning (UIC) April 10, 2025

Lecture 21

Instructor: Aadirupa Saha *Scribe(s): (Haoxuan Wang)*

Overview

In the last lecture, we covered the following main topics:

1. PCA
2. Eigen vector/value calculation

This lecture focuses on:

1. K-Means clustering
2. Cluster number determination
3. Kernelized K-Means
4. Spectral Clustering
5. DBSCAN (advanced)

1 Clustering

Clustering is an **unsupervised learning** technique to "group" a subset of instances.

More technically:

Let

$$\mathcal{D} = \{x_i\}_{i=1}^{n}$$

be a dataset consisting of instances.
Goal: Find "separated" partitions of $\mathcal{D}$ such that:

$$\mathcal{D} = D_1 \cup D_2 \cup D_3 \cup \cdots \cup D_K, \quad [K = \text{\# clusters}]$$

subject to:

$$D_i \cap D_j = \emptyset \quad \forall i \neq j \in [K]$$

This is called **hard clustering**.

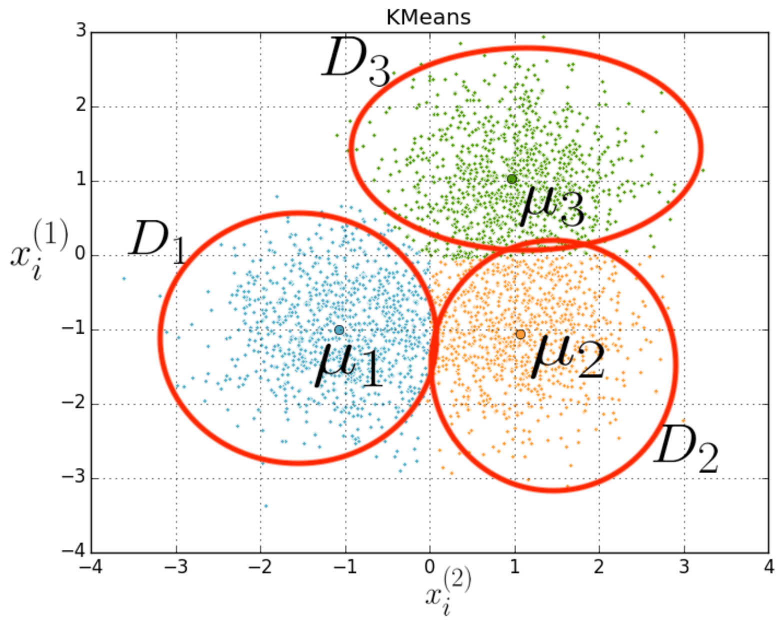

Figure 1: A 2-D clustering example.

Example:

As shown in Fig. 1, if $d = 2$ (2-dimensional dataset), and

$$\mathcal{D} = \left\{ \begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \end{pmatrix}, \begin{pmatrix} x_1^{(2)} \\ x_2^{(2)} \end{pmatrix}, \ldots, \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \end{pmatrix} \right\}, \quad \text{with } K = 3$$

Each cluster D_1, D_2, D_3 has its own center μ_1, μ_2, μ_3 respectively.

Clustering Assignment Notation

We denote the clustering assignment as a mapping:

$$C : [n] \to [k],$$

where $C(i)$ is the cluster index assigned to data point i.

- For any cluster $c \in [k]$, we also denote by $\mu_c \in \mathbb{R}^d$ the cluster centroid / head.
- Every partition-c for the c-th cluster is denoted by:

$$D_c = \{i \in [n] \mid C(i) = c\}, \quad \text{for } c \in [k].$$

2 K-Means Clustering

2.1 Supervised Learning and Unsupervised Learning

Supervised learning (SL) refers to learning a mapping from an input space $X \subset \mathbb{R}^d$ to an output space Y. The training data set is

$$D_{\text{SL}} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\},$$

where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)}$ is the label or target value (e.g., a class for classification, or a real value for regression). The goal is to learn a function

$$f : X \to Y$$

that predicts y accurately for new inputs x. Examples of supervised learning tasks:

- *Classification:* $Y = \{1, 2, \ldots, K\}$ or $\{\texttt{cat}, \texttt{dog}, \ldots\}$
- *Regression:* $Y \subseteq \mathbb{R}$

Unsupervised learning (USL) deals with unlabeled data. Here, the training set is

$$D_{\text{USL}} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}, \quad x^{(i)} \in \mathbb{R}^d,$$

with no corresponding labels $y^{(i)}$. The goal often involves discovering hidden structure or patterns in the data. *Clustering* is a key example, in which we seek to partition $\{x^{(i)}\}$ into groups (clusters) such that points within each group are more similar to each other than to those in other groups.
Formally, one can define a clustering function

$$G : X \to \{1, 2, \ldots, K\},$$

where $G(x^{(i)}) \in \{1, 2, \ldots, K\}$ is the cluster index assigned to point $x^{(i)}$. Equivalently, we can define clusters $C_k \subset \{1, 2, \ldots, n\}$ as

$$C_k = \{i \mid G(x^{(i)}) = k\}, \quad k = 1, \ldots, K.$$

2.2 K-Means Clustering

K-Means is a classic clustering algorithm aiming to partition n points into K clusters. Intuitively, each cluster is represented by a *centroid*, and each data point is assigned to the cluster whose centroid is closest in terms of Euclidean distance.

Objective Function:

Let

$$\mu_k \in \mathbb{R}^d \quad (k = 1, \ldots, K)$$

denote the centroid of cluster C_k. K-Means minimizes the sum of squared distances between each data point and its assigned cluster centroid:

$$J(\mu_1, \ldots, \mu_K, C_1, \ldots, C_K) = \sum_{k=1}^{K} \sum_{x^{(i)} \in C_k} \|x^{(i)} - \mu_k\|^2.$$

Algorithm Description:

K-Means proceeds iteratively, alternating between an *assignment step* and an *update step*:

1. **Initialization:** Pick K initial centroids $\mu_1^{(0)}, \mu_2^{(0)}, \ldots, \mu_K^{(0)}$. Sometimes this is done at random, or by a more refined method such as *K-Means++*.
2. **Assignment step:** For each point $x^{(i)}$, assign it to the cluster whose centroid is closest:
$$C_k^{(t)} = \{x^{(i)} \mid k = \arg\min_j \|x^{(i)} - \mu_j^{(t)}\|^2\}.$$
3. **Update step:** Recompute each centroid as the mean of points in the corresponding cluster:
$$\mu_k^{(t+1)} = \frac{1}{|C_k^{(t)}|} \sum_{x^{(i)} \in C_k^{(t)}} x^{(i)}.$$
4. **Convergence check:** Repeat the above assignment and update steps until the centroids stabilize or a maximum number of iterations is reached.

Algorithm 2.1: K-Means Algorithm

Input:
$$\mathcal{D} = \{x_1, \ldots, x_n\}, \quad k \text{ (number of clusters)}$$
Initialization: Let C^0 be some arbitrary clustering assignment. Let $\mu_1, \mu_2, \ldots, \mu_k$ be any arbitrary points in $\mathcal{D}$ (initial centroids). `Flag` = true
while `Flag` is true **do**
    Set `Flag` = false
    for all $i \in [n]$ **do** If $\exists\, \tilde{k} \in [k]$ such that $C^t(i) \neq \tilde{k}$ **but**
$$\|x_i - \mu_{\tilde{k}}^t\|_2^2 < \|x_i - \mu_{C^t(i)}^t\|_2^2,$$
        then set $C^{t+1}(i) \leftarrow \tilde{k}$, `Flag` = true
    end for
    Update cluster centroids:
$$\mu_{\tilde{k}}^{t+1} = \frac{1}{|D_{\tilde{k}}^t|} \sum_{i \in D_{\tilde{k}}^t} x_i, \quad \forall\, \tilde{k} \in [k]$$
    where
$$D_{\tilde{k}}^t = \left\{ i \in [n] \;\middle|\; C^t(i) = \tilde{k} \right\} \quad \text{is the partition for the } \tilde{k}\text{-th cluster.}$$
    Increment $t \leftarrow t + 1$
end while
Output: Return final clustering assignment C^{t+1} and centroids $(\mu_1^{t+1}, \ldots, \mu_k^{t+1})$.

More analysis on the algorithm

- **Convergence:** K-Means always terminates in a finite number of iterations (each iteration strictly decreases the objective), although it may converge to a local rather than global minimum.
- **Initialization Sensitivity:** The algorithm's final solution depends on the initial centroids; multiple runs or K-Means++ can mitigate poor initialization.
- **Complexity:** Each iteration takes $O(nKd)$ time (for n points, K centroids, and dimension d). Over I iterations, total complexity is $O(nKdI)$.
- **Choosing K:** Often done via heuristic (e.g. the *elbow method*), domain knowledge, or more advanced methods like silhouette analysis.

2.3 Convergence of K-Means

Definition: Sum of Squared Errors (SSE)

One common way to quantify the *loss* or *cost* for a clustering algorithm is via the **Sum of Squared Errors / Sum of Squared Distances** (SSE/SSD). Let

$$\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\} \subset \mathbb{R}^d$$

be our dataset, given a clustering mapping C^t and corresponding centroids μ^t, the objective function is defined as:

$$\text{SSE}(C^t, \mu^t) = \sum_{i=1}^{n} \left\| x_i - \mu^t_{C^t(i)} \right\|^2$$

This measures the sum of squared distances of data points from their assigned cluster centers.

Theorem 1: K-Means Algorithm Converges

$$\forall t: \quad \text{SSE}(C^{t+1}, \mu^{t+1}) < \text{SSE}(C^t, \mu^t)$$

Proof:

The proof proceeds in two steps:

- **Step 1:**
$$\text{SSE}(C^{t+1}, \mu^t) < \text{SSE}(C^t, \mu^t)$$
- **Step 2:**
$$\text{SSE}(C^{t+1}, \mu^{t+1}) < \text{SSE}(C^{t+1}, \mu^t)$$

Proof of Step 1:

This follows directly from the update step of K-Means:

$$\text{LHS} = \text{SSE}(C^{t+1}, \mu^t) = \sum_{i=1}^{n} \left\| x_i - \mu^t_{C^{t+1}(i)} \right\|^2$$
$$< \sum_{i=1}^{n} \left\| x_i - \mu^t_{C^t(i)} \right\|^2 = \text{SSE}(C^t, \mu^t) = \text{RHS}$$

Proof of Step 2:

This step uses the fact that updating centroids minimizes SSE over fixed assignments C^{t+1}.

Lemma 1

Consider points $z'_1, z'_2, \ldots, z'_m \in \mathbb{R}^d$ *where* $m \geq 1$, and let

$$z^* = \frac{1}{m} \sum_{i=1}^{m} z'_i$$

be any point in the same d-dimensional space. Then:

$$\sum_{i=1}^{m} \|z'_i - z\|^2 \geq \sum_{i=1}^{m} \|z'_i - z^*\|^2 \quad \text{for any } z \in \mathbb{R}^d.$$

Proof of Lemma 1

Define:

$$f(z) := \sum_{i=1}^{m} \|z'_i - z\|^2, \quad \text{for any } z \in \mathbb{R}^d.$$

- $f(z)$ is convex.
- Gradient:
$$\nabla f(z) = -2 \sum_{i=1}^{m} (z'_i - z)$$
- Setting $\nabla f(z) = 0$ yields:
$$m z^* = \sum_{i=1}^{m} z'_i \quad \Rightarrow \quad z^* = \left(\sum_{i=1}^{m} z'_i \right) \Big/ m.$$

Thus, the function $f(z)$ is minimized at $z^* = \frac{1}{m} \sum_{i=1}^{m} z'_i$, completing the proof. $\square$

Continuing the Proof of Step 2

By Lemma 1:

$$\text{SSE}(C^{t+1}, \mu^{t+1}) = \sum_{i=1}^{n} \left\| x_i - \mu^{t+1}_{C^{t+1}(i)} \right\|^2 = \sum_{k'=1}^{k} \sum_{i \in D^{t+1}_{k'}} \left\| x_i - \mu^{t+1}_{k'} \right\|^2$$

Using Lemma 1:

$$\leq \sum_{k'=1}^{k} \sum_{i \in D^{t+1}_{k'}} \left\| x_i - \mu^{t}_{k'} \right\|^2 = \text{SSE}(C^{t+1}, \mu^{t})$$

Conclusion

The proof of Theorem 1 (K-Means convergence) follows by combining the results of **Step 1** and **Step 2**.

3 Choosing the Number of Clusters K

How to Choose k?

(B) Silhouette Scoring Method

When applying K-Means clustering, one of the practical challenges is selecting an appropriate number of clusters K. Two common heuristic approaches are the **Elbow Method** and the **Silhouette Method**.

3.1 The Elbow Method

Let $\text{SSE}(K)$ be the sum of squared errors achieved by K-Means when we choose K clusters (also called *within-cluster sum of squares*, WCSS). Formally:

$$\text{SSE}(K) \;=\; \sum_{k=1}^{K} \sum_{x^{(i)} \in C_k} \left\| x^{(i)} - \mu_k \right\|^2.$$

We compute $\text{SSE}(K)$ for a range of values $K = 1, 2, \ldots, K_{\max}$ (e.g. up to some reasonable upper bound). Plotting $\text{SSE}(K)$ as a function of K typically yields a monotonically decreasing curve; as K increases, $\text{SSE}(K)$ generally decreases (since more clusters can capture finer distinctions).

$$\begin{array}{c} \text{SSE}(K) \\ \big\downarrow \\ \text{(Elbow shape)} \end{array} \qquad K = 1 \quad \to \quad K = 2 \quad \to \quad \ldots \quad \to \quad K_{\max}$$

Elbow Criterion. Look for a point $K = k^*$ on this curve where the rate of decrease (i.e., the slope) significantly changes (the *knee* or *elbow*). This indicates that increasing K beyond k^* yields diminishing returns in lowering the SSE, suggesting k^* is a good choice.

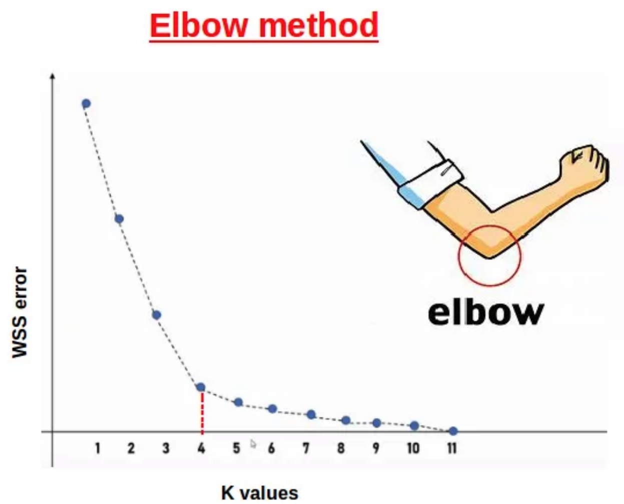

Figure 2: Illustrations of the Elbow Method (Ref. 4). The Y-axis (scoring metric) may vary based on the specific problems.

3.2 The Silhouette Method

Another popular approach is to use the **Silhouette Score**, which quantifies how well each data point "fits" within its assigned cluster compared to other clusters. Let

$$a(i) \;=\; \text{average distance of } x^{(i)} \text{ to other points in the same cluster,}$$

$$b(i) \;=\; \min_{k \neq c(i)} \big\{\text{average distance of } x^{(i)} \text{ to the points in cluster } k\big\},$$

where $c(i)$ denotes the cluster index assigned to $x^{(i)}$. The *silhouette score* $s(i)$ for point $x^{(i)}$ is:

$$s(i) \;=\; \frac{b(i) - a(i)}{\max\big\{a(i),\, b(i)\big\}}.$$

- If $s(i)$ is close to $+1$, then $x^{(i)}$ is far from the other clusters but tightly grouped within its own cluster (good).
- If $s(i)$ is close to 0, then $x^{(i)}$ lies on or near a cluster boundary.
- If $s(i)$ is negative, then $x^{(i)}$ is possibly assigned to the wrong cluster.

Overall Silhouette Score. We then take the average silhouette score over all points:

$$S = \frac{1}{n}\sum_{i=1}^{n} s(i).$$

For each candidate K, we run K-Means and compute S. A higher average silhouette score indicates better-defined clusters. We choose

$$K = \arg\max_{K} S.$$

Often one plots the average silhouette score versus K and picks the value of K that yields the highest peak or a suitably high silhouette score.

Note:

$$S \in [-1, 1],$$

and a value of 1 indicates a "perfect" clustering assignment.

The heuristic would be to select k^* with sufficiently high silhouette score.

Summary

- **Elbow Method:** Look for the "knee" in the plot of SSE(K) vs. K.
- **Silhouette Method:** Compute silhouette scores for each K and pick the K with the highest average silhouette score.

These are heuristic methods—there is no absolute guarantee that the chosen K is optimal, but they serve as practical, widely-used guidelines in real-world applications.

Code example

Silhouette Method

4 Kernelized K-Means

Kernelized K-Means extends the classical K-Means algorithm by mapping the original data into a high-dimensional feature space via a non-linear function, and then performing clustering in that space. This enables the algorithm to discover clusters that are non-linearly separable in the original space.

4.1 Motivation and Objective

Given data points

$$\mathcal{X} = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\} \subset \mathbb{R}^d,$$

we wish to partition them into K clusters. In kernelized K-Means, a mapping $\varphi : \mathbb{R}^d \to \mathcal{H}$ is used to transform each $x^{(i)}$ into a feature space $\mathcal{H}$. The clustering objective in the feature space is given by:

$$J = \sum_{k=1}^{K} \sum_{x^{(i)} \in C_k} \left\| \varphi(x^{(i)}) - \mu_k \right\|^2,$$

where the centroid μ_k for cluster C_k is computed as:

$$\mu_k = \frac{1}{|C_k|} \sum_{x^{(i)} \in C_k} \varphi(x^{(i)}).$$

4.2 Kernel Trick and Distance Computation

Rather than computing $\varphi(x)$ explicitly, we use a kernel function $k(x, y) = \langle\varphi(x), \varphi(y)\rangle$ to determine distances in $\mathcal{H}$. The squared distance between $\varphi(x^{(i)})$ and the centroid μ_k is:

$$\begin{aligned}\left\|\varphi(x^{(i)}) - \mu_k\right\|^2 &= \langle\varphi(x^{(i)}), \varphi(x^{(i)})\rangle - \frac{2}{|C_k|}\sum_{x^{(j)}\in C_k}\langle\varphi(x^{(i)}), \varphi(x^{(j)})\rangle + \frac{1}{|C_k|^2}\sum_{x^{(j)},x^{(l)}\in C_k}\langle\varphi(x^{(j)}), \varphi(x^{(l)})\rangle \\ &= k(x^{(i)}, x^{(i)}) - \frac{2}{|C_k|}\sum_{x^{(j)}\in C_k} k(x^{(i)}, x^{(j)}) + \frac{1}{|C_k|^2}\sum_{x^{(j)},x^{(l)}\in C_k} k(x^{(j)}, x^{(l)}).\end{aligned}$$

4.3 Algorithm Outline

Kernelized K-Means follows a similar iterative procedure as the standard algorithm:

1. **Initialization:** Choose K initial clusters (or centroids in the feature space) by selecting initial points or using a method like K-Means++.

2. **Assignment Step:** For each point $x^{(i)}$, assign it to the cluster that minimizes the kernel-based squared distance.

3. **Update Step:** Recompute the cluster "centroids" implicitly by using the kernel values:

   $$\mu_k \leftarrow \frac{1}{|C_k|}\sum_{x^{(i)}\in C_k}\varphi(x^{(i)}).$$

   Note that the centroids are not computed explicitly; only distances (expressed in terms of $k(x, y)$) are needed.

4. **Convergence Check:** Repeat the assignment and update steps until the objective function J converges or changes minimally.

Algorithm 4.1: Kernelized K-Means Clustering Algorithm

Input: Data set $\{x^{(1)}, \ldots, x^{(n)}\}$, number of clusters K, kernel function $k(\cdot, \cdot)$.
Output: Clusters $C_1, C_2, \ldots, C_K$
Assign initial cluster memberships for each $x^{(i)}$.
while not convergence or not reach maximum iterations **do**
  for each $x^{(i)}$ **do**
    Compute squared distances using

$$d^2(x^{(i)}, C_k) = k(x^{(i)}, x^{(i)}) - \frac{2}{|C_k|}\sum_{x^{(j)}\in C_k} k(x^{(i)}, x^{(j)}) + \frac{1}{|C_k|^2}\sum_{x^{(j)},x^{(l)}\in C_k} k(x^{(j)}, x^{(l)})$$

    Assign $x^{(i)}$ to the cluster with the minimum computed distance.
  end for
  Update cluster memberships based on new assignments.
end while

Remarks: Kernelized K-Means can capture complex, non-linear cluster boundaries, but the choice of kernel (e.g., Gaussian, polynomial) is crucial to its performance.

5 Spectral Clustering (advanced clustering method)

Spectral Clustering provides a powerful method to partition data (or the nodes of a graph) into clusters by leveraging the eigenstructure of a *graph Laplacian* derived from the data. It is particularly well-suited for non-convex or "manifold-like" structures that can be difficult for algorithms like K-Means to detect.

- **Goal:** Separate data into groups (clusters) such that points within the same cluster are highly "connected," and points in different clusters are less connected.
- **Key idea:** Encode the data as a graph, construct a suitable *Laplacian matrix*, then use its eigenvectors to embed data into a new space where a simple clustering method (like K-Means) suffices.

5.1 From Data to Graphs

Adjacency Matrix W

Given n data points $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\} \subset \mathbb{R}^d$, we build an undirected similarity graph $G = (V, E)$ with:

$$V = \{1, 2, \ldots, n\} \quad \text{(one vertex per data point)},$$

and an adjacency matrix

$$W \in \mathbb{R}^{n \times n}, \quad W_{ij} \geq 0.$$

Several common ways to define W:

- k**-Nearest Neighbor Graph:** Connect each point to its k nearest neighbors in feature space. Then $W_{ij} = 1$ (or a weighted value) if $x^{(j)}$ is among the k neighbors of $x^{(i)}$, else 0.
- ε**-Neighborhood Graph:** Connect $x^{(i)}$ and $x^{(j)}$ if $\|x^{(i)} - x^{(j)}\| \leq \varepsilon$. Then $W_{ij} = 1$ (or a function of the distance) if they are within ε, else 0.
- **Heat Kernel / Gaussian Similarity:**

$$W_{ij} = \exp\Big(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\Big) \quad \text{if } x^{(i)} \text{ and } x^{(j)} \text{ are neighbors (in one of the above senses).}$$

Degree Matrix D

Define the **degree** of each node i by summing up all its edge weights:

$$D_{ii} = \sum_{j=1}^{n} W_{ij}, \quad \text{and } D_{ij} = 0 \text{ for } i \neq j.$$

Hence, D is a diagonal matrix with entries $\{D_{11}, D_{22}, \ldots, D_{nn}\}$.

5.2 Graph Laplacians

Unnormalized Laplacian L

The *unnormalized Laplacian* is defined as

$$L = D - W.$$

Key properties:

- L is symmetric and positive semi-definite.
- The smallest eigenvalue of L is 0, with corresponding eigenvector $\mathbf{1} = (1, 1, \dots, 1)^\top$.
- For partitioning into two sets, the second smallest eigenvector (the *Fiedler vector*) often provides valuable information.

Normalized Laplacians L_{sym} and L_{rw}

Sometimes, it is advantageous to normalize the Laplacian to account for uneven degrees. Two common variants:

$$L_{\text{sym}} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}},$$
$$L_{\text{rw}} = D^{-1}(D - W) = I - D^{-1}W.$$

- L_{sym} is symmetric, which is often convenient for analysis.
- L_{rw} can be interpreted in terms of random walks on the graph.

5.3 Normalized Cuts

Spectral clustering can be seen as an approach to approximately minimize the **Normalized Cut (Ncut)** of the graph. For a bipartition (S, T) of the vertices:

$$\text{cut}(S, T) = \sum_{i \in S,\, j \in T} W_{ij},$$

$$\text{Ncut}(S, T) = \frac{\text{cut}(S, T)}{\text{assoc}(S, V)} + \frac{\text{cut}(S, T)}{\text{assoc}(T, V)},$$

where

$$\text{assoc}(S, V) = \sum_{i \in S,\, j \in V} W_{ij}.$$

Minimizing Ncut can be related to the eigenvectors of L_{sym} or L_{rw}. For multiway partitions (more than 2 clusters), the approach extends by considering multiple eigenvectors.

5.4 Spectral Clustering Algorithms

Below is a simplified outline for **normalized spectral clustering** using L_{sym}. (Unnormalized and random-walk versions are conceptually similar, with minor algebraic differences.)

1. **Form the Similarity Graph.**
   - Construct W using, e.g., k-nearest neighbors or ε-neighborhoods with a suitable kernel.
   - Compute the degree matrix D.
2. **Compute the Laplacian.**
$$L_{\text{sym}} = I - D^{-\frac{1}{2}}\, W\, D^{-\frac{1}{2}}.$$
3. **Compute the First K Eigenvectors.**
   - Let $\mathbf{u}_1, \dots, \mathbf{u}_K$ be the eigenvectors of L_{sym} corresponding to the *smallest* K eigenvalues (the "bottom" K eigenvectors).
4. **Form Embeddings and Normalize Rows.**
   - Let $U \in \mathbb{R}^{n \times K}$ be the matrix with columns $\mathbf{u}_1, \dots, \mathbf{u}_K$.
   - Normalize each row of U to unit length to get T. (In some versions, this step may vary.)
5. **Cluster the Rows of T.**
   - Each row of T is now a K-dimensional embedding of the original vertex $x^{(i)}$.
   - Apply K-Means (or another clustering method) to the rows of T to partition the data into K clusters.
6. **Assign Original Points.**
   - Let the resulting clusters from the K-Means step define the final clusters of the graph vertices.

5.5 Interpretation and Remarks

- **Manifold vs. Graph:** Even if the data lie on a high-dimensional manifold, constructing a graph and using the Laplacian's spectral properties captures non-linear structures more readily than linear methods such as PCA.
- **Choice of K:** As with other clustering approaches, one may use the Elbow Method, Silhouette Scores, or domain knowledge to pick K.
- **Complexity:** The most computationally expensive step is the eigen-decomposition, typically $O(n^3)$ in the worst case. For large-scale problems, approximate or sparse methods are used.
- **Normalized vs. Unnormalized:** Normalized Laplacians often yield more robust results, especially if node degrees vary widely.

6 DBSCAN (advanced clustering method)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm designed to discover clusters of arbitrary shape and to identify noise/outlier points. Unlike K-Means, DBSCAN does not require specifying the number of clusters in advance.

6.1 Key Concepts and Definitions

- **ε-Neighborhood:** For a point $x^{(i)}$, the ε-neighborhood is defined as:

$$N_\varepsilon(x^{(i)}) = \{x^{(j)} \mid \|x^{(i)} - x^{(j)}\| \leq \varepsilon\}.$$

- **MinPts:** The minimum number of points required to form a dense region.
- **Core Point:** A point $x^{(i)}$ is a core point if $|N_\varepsilon(x^{(i)})| \geq \text{MinPts}$.
- **Border Point:** A point that is not a core point but falls within the ε-neighborhood of a core point.
- **Noise Point:** A point that is neither a core point nor a border point.

6.2 Algorithm Description

DBSCAN clusters points based on density connectivity:

1. **Randomly select a point $x^{(i)}$.** If it has not been visited, mark it as visited.
2. **Retrieve the ε-neighborhood $N_\varepsilon(x^{(i)})$.**
3. **Core Point Check:** If $|N_\varepsilon(x^{(i)})| < \text{MinPts}$, mark $x^{(i)}$ as noise; otherwise, start a new cluster.
4. **Cluster Expansion:** If $x^{(i)}$ is a core point, recursively add all points in $N_\varepsilon(x^{(i)})$ that meet the density criteria to the cluster. Continue this process for each neighbor point that is a core point.
5. **Iterate:** Continue until all points have been visited.

Algorithm 6.1: DBSCAN Clustering Algorithm

Input: Data set $\{x^{(1)}, \dots, x^{(n)}\}$, parameters ε and MinPts.
Output: Clustered data points, with noise points identified.
Initialize all points as *unvisited*.
for each point $x^{(i)}$ **do**
  if $x^{(i)}$ is not visited **then**
    Mark $x^{(i)}$ as visited
    $N \leftarrow N_\varepsilon(x^{(i)})$
    if $|N| <$ MinPts **then**
      Mark $x^{(i)}$ as noise
    else
      Create a new cluster C and add $x^{(i)}$
      ExpandCluster($x^{(i)}, N, C$ **)**
    end if
  end if
end for

Algorithm 6.2: ExpandCluster Function in DBSCAN

for each point $x' \in N$ **do**
  if x' is not visited **then**
    Mark x' as visited
    $N' \leftarrow N_\varepsilon(x')$
    if $|N'| \geq$ MinPts **then**
      $N \leftarrow N \cup N'$
    end if
  end if
  if x' is not yet part of any cluster **then**
    Add x' to cluster C
  end if
end for

6.3 Intuition and Discussion

- **Cluster Shape:** DBSCAN can find clusters of arbitrary shape since it groups points based on local density rather than relying on a global distance metric, allowing for the discovery of clusters with arbitrary shapes and automatically identifying noise.
- **No Need to Specify K:** Unlike K-Means, DBSCAN does not require the number of clusters to be known beforehand.
- **Handling Noise:** Points that do not belong to any cluster (i.e., those in sparse regions) are naturally labeled as noise.
- **Parameter Sensitivity:** The choice of ε and MinPts is crucial. Too small ε may lead to many small clusters or label most points as noise, whereas too large ε may merge distinct clusters.

Next Lecture

The next lecture will cover the following topics:
(i) Neural Nets
(ii) Back Propagation

References:

1. Pattern Recognition and Machine Learning by Christopher Bishop, page 424.
2. k-means Clustering by Shivaram Kalyanakrishnan. Link.
3. Survey of clustering algorithms, Rui Xu; D. Wunsch. Link
4. A Tutorial on Spectral Clustering, by Ulrike von Luxburg. Link
5. Online blogs: Link 1, Link 2
6. ChatGPT, OpenAI