Spectral Clustering for Unsupervised Learning

Numerical Methods for Deep Learning

Motivation: Data Mining

Assume we have data

$$\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n]$$

.

The data can be

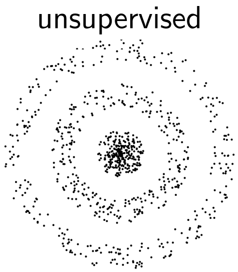
- Images
- Text
- Sound
- Set of numbers (climate, pressure ...)

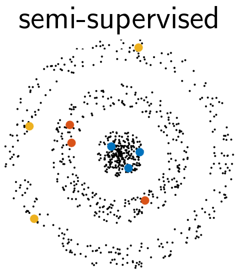
We can think of (at least) three goals:

- Cluster the data (unsupervised)
- Give meaning to each cluster, label it (semisupervised)
- Find a functional relation between the cluster and its label (supervised)

Example: Un/Semi/Fully Supervised Learning

Example: $\mathbf{y} \in \mathbb{R}^2$ contains rock conductivity porosity





- How many types of rocks do we have? **Unsupervised learning**
- What are their names (Granite, Basalt)? **Semisupervised learning**
- Given σ, ϕ can we find a function $f(\sigma, \phi) = \mathrm{rock\ type}$? **Supervised learning**

Discussion: Un/Semi/Fully Supervised Learning

- Supervised learning requires a large labeled data set
- Gives an "explanation" (a model) between data and label
- Un/Semisupervised is more modest
- No model, just label
- Can be followed by supervised learning

Learning Objective: Spectral Clustering

In this module we focus on spectral clustering algorithms for unsupervised and semisupervised learning.

Learning tasks:

- Given a data set cluster the data into a few groups
- Assuming that a few data are labeled, label the rest

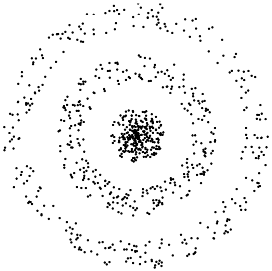
Numerical methods:

- k-means algorithm (not discussed)
- spectral clustering (our focus)

Spectral Clustering: General Idea

Idea: Given the data set $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n]$, if $\mathbf{y}_i$ is similar to $\mathbf{y}_j$ then they belong to the same class.

Central question: How to measure and use similarity?



Literature: [1, 2, 3]

Similarity Measures: Metrics - 1

How to quantify the similarity of two data points?

A function $D : \mathbb{R}^{n_f} \times \mathbb{R}^{n_f} \to [0, \infty)$ is called a *metric*, if for all $\mathbf{x}, \mathbf{y}, \mathbf{z}$ it holds that

- $D(\mathbf{x}, \mathbf{y}) \geq 0$
- $D(\mathbf{x}, \mathbf{y}) = 0 \quad \Rightarrow \quad \mathbf{x} = \mathbf{y}$
- $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$
- $D(\mathbf{x}, \mathbf{z}) \leq D(\mathbf{x}, \mathbf{y}) + D(\mathbf{y}, \mathbf{z})$

A metric can be used to measure similarity.

Note: Not all properties are needed. E.g., sometimes we will drop the second one.

Similarity Measures: Metrics - 2

Most obvious metric is induced by a norm, e.g.,

$$D(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p = \left(\sum_{i=1}^{n_f} |\mathbf{x}_i - \mathbf{y}_i|^p \right)^{\frac{1}{p}}$$

- most used is the 2-norm (why?)
- $p = \infty$ is called the max norm (why?)

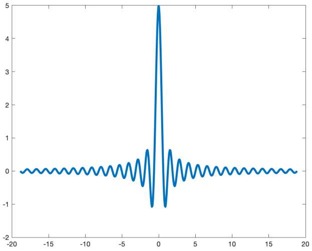
A simple modification: weighted norms

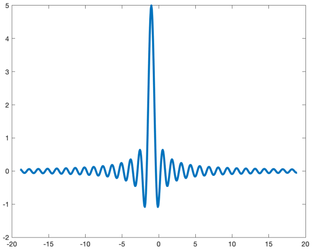
$$D(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n_f} \mathbf{w}_i |\mathbf{x}_i - \mathbf{y}_i|^p \right)^{\frac{1}{p}}$$

where $\mathbf{w} > 0$ is a vector of positive weights. Allows us to focus on some components and scale the metric.

Similarity Measures: Metrics - 3

Are these signals similar?





Can we find a translation-invariant metric?

Similarity Measures: Metrics - 4

Which of these image pairs are most similar?

(by Christmas w/a K)

(by Art Bromage)

(by P R Simões)

Similarity Measures: Metrics - 5

Many different definitions of distance based on the application

- normed distance
- Hamming distance - for strings
- Wasserstein metric - for probabilities (also applied to images)
- Riemannian metrics - for data that "lives" on manifolds

Choosing the right similarity measure is the key for the application.
In many cases - chicken and egg. If we know the right distance then we know how to cluster.

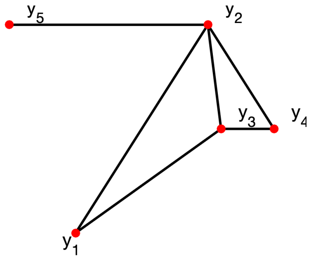
We will use $\| \cdot \|_2$ for simplicity.

Undirected Graphs

The graph $G(\mathbf{Y}, \mathcal{E})$ is described by its vertices $\mathbf{y}_1, \dots, \mathbf{y}_n$ and an edge set $\mathcal{E}$.

Example:

```
Y = [-1 0 .1 .5 -1.5; -1 1 0 0 1]
E = [1 2;1 2;2 3;1 3;3 4;2 4;2 5]
```

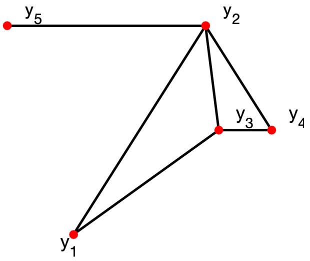


We will only use undirected graphs: $(i, j) \in \mathcal{E} \Rightarrow (j, i) \in \mathcal{E}$.

Adjacency Matrix of a Graph

The adjacency matrix of $G(\mathbf{Y}, \mathcal{E})$ is $\mathbf{A} \in \{0,1\}^{n \times n}$ with

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if there is an edge between } \mathbf{y}_i \text{ and } \mathbf{y}_j \\ 0, & \text{else.} \end{cases}$$



$$\Rightarrow \mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

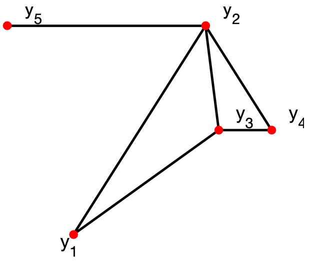
Note: $\mathbf{Y}$ and $\mathbf{A}$ fully describe the graph (no need to store $\mathcal{E}$).

Degree Matrix of a Graph

The degree matrix is

$$\mathbf{D}_{ii} = \mathrm{diag}\left(\sum_j \mathbf{A}_{ij}\right) \in \mathbb{R}^{n \times n}$$

.



$$\Rightarrow \mathbf{D} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$\mathbf{D}_{ii}$ is the degree of node $\mathbf{y}_i$ ($\approx$ importance of $\mathbf{y}_i$).

Weighted Adjacency Matrix of a Graph

The weighted adjacency matrix of $G(\mathbf{Y}, \mathcal{E})$ is $\mathbf{W} \in \mathbb{R}_+^{n \times n}$ with

$$\mathbf{W}_{ij} \begin{cases} > 0 & \text{if } \mathbf{A}_{ij} = 1 \\ = 0 & \text{otherwise.} \end{cases}$$

Use a similarity measure to compute the entries in $\mathbf{W}$, e.g.,

$$\mathbf{W}_{ij} = \exp\left(-\frac{D(\mathbf{y}_i, \mathbf{y}_j)}{\sigma}\right).$$

As before, the degree matrix is

$$\mathbf{D}_{ii} = \mathrm{diag}\left(\sum_j \mathbf{W}_{ij}\right) \in \mathbb{R}^{n \times n}.$$

$\mathbf{D}_{ii}$ is the degree of node $\mathbf{y}_i$ ($\approx$ importance of $\mathbf{y}_i$).

k-Nearest Neighbor Graph

Given: Data $\mathbf{Y}$, $k \in \mathbb{N}$, and a distance function D.
Idea: Set $\mathbf{A}_{ij} = \mathbf{A}_{ji} = 1$ if $\mathbf{y}_i$ is among the k nearest neighbors of $\mathbf{y}_j$ with respect to D or vice versa.

This is not the only option. Common alternatives:

- **ϵ-neighborhood graph:** add an edge between $\mathbf{y}_i$ and $\mathbf{y}_j$ if $D(\mathbf{y}_i, \mathbf{y}_j) < \epsilon$
- **mutual k-nearest neighborhood graph:** add an edge between $\mathbf{y}_i$ and $\mathbf{y}_j$ if $\mathbf{y}_i$ is among the k nearest neighbors of $\mathbf{y}_j$ *and* vice versa.
- **fully-connected graph:** add an edge between all examples.

In all these cases, we return the weighted adjacency matrix, whose entries depend on D.

The Graph Laplacian

The graph Laplacian of $G(\mathbf{Y}, \mathcal{E})$ is

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

For our example, we get

$$\mathbf{L} = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 4 & -1 & 0 \\ 0 & -1 & -1 & 2 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{pmatrix}$$

Note that for a vertex function $\mathbf{v} \in \mathbb{R}^n$

$$(\mathbf{L}\mathbf{v})_i = \sum_{j=1}^{n} \mathbf{A}_{ij}(\mathbf{v}_i - \mathbf{v}_j) \quad \text{and} \quad \mathbf{v}^\top \mathbf{L}\mathbf{v} = \sum_{i,j=1}^{n} \mathbf{A}_{ij}(\mathbf{v}_i - \mathbf{v}_j)^2$$

Weighted Graph Laplacian

Similar to the previous slide, we define the weighted graph Laplacian of $G(\mathbf{Y}, \mathcal{E})$ as

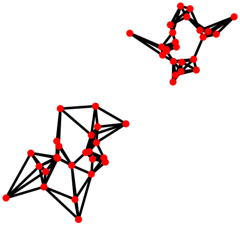
$$\mathbf{L} = \mathbf{D} - \mathbf{W}$$

($\mathbf{D}$ is the degree matrix of the weighted adjacency matrix $\mathbf{W}$)

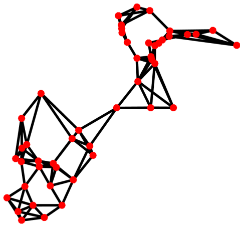
For a vertex function $\mathbf{v} \in \mathbb{R}^n$, we obtain

$$(\mathbf{L}\mathbf{v})_i = \sum_{j=1}^{n} \mathbf{W}_{ij}(\mathbf{v}_i - \mathbf{v}_j) \quad \text{and} \quad \mathbf{v}^\top \mathbf{L}\mathbf{v} = \sum_{i,j=1}^{n} \mathbf{W}_{ij}(\mathbf{v}_i - \mathbf{v}_j)^2$$

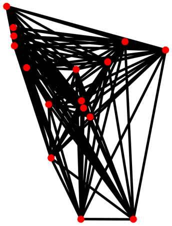
Eigenvalues of the Graph Laplacian



$\lambda_2 \approx 1.9 \cdot 10^{-15}$



$\lambda_2 \approx 8.7 \cdot 10^{-3}$



$\lambda_2 \approx 4.0 \cdot 10^{-1}$

The first (i.e., smallest) eigenvalue is always 0 with eigenvector $\mathbf{v} = [1, \dots, 1]^\top$ (why?)

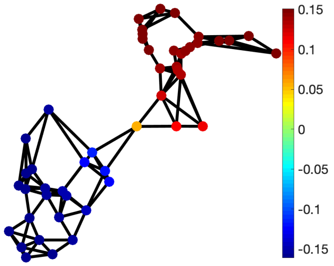
The multiplicity of the eigenvalue 0 equals the number of connected components in the graph.

The smallest non-zero eigenvalue is the Fiedler eigenvalue.

Fiedler Vector

Assume there is only one connected component. Then, the eigenvector associated with the second eigenvalue can be obtained by solving

$$\begin{aligned} \min_{\mathbf{u} \neq \gamma \mathbf{e}} \quad & \frac{1}{2}\mathbf{u}^\top \mathbf{L}\mathbf{u} \\ \text{subject to} \quad & \|\mathbf{u}\| = 1 \end{aligned}$$



Interpretation: Find the non-constant vector with the minimal energy (i.e., as smooth as possible)

It (approximately) holds that $\mathbf{u}_i > 0$ for examples in the first group and $\mathbf{u}_i < 0$ for examples in the second group.

Computing the Fiedler Vector

The Fiedler vector can be computed by solving the optimization problem

$$\begin{aligned} \min_{\mathbf{u} \neq \gamma \mathbf{e}} \quad & \frac{1}{2}\mathbf{u}^\top \mathbf{L}\mathbf{u} \\ \text{subject to} \quad & \|\mathbf{u}\| = 1 \end{aligned}$$

General idea:

- For small problems use eig solver of a dense matrix
- For large problem use iterative methods - inverse iteration, Krylov methods, randomized linear algebra

Simple (but not very efficient) option is the power method:

$$\mathbf{v}_k = \mathbf{L}^\dagger \mathbf{u}_k \qquad \mathbf{u}_{k+1} = \frac{\mathbf{v}_k}{\|\mathbf{v}_k\|}, \quad k = 1, \ldots$$

Algorithm: Normalized Spectral Clustering [1]

Input: Weighted adjacency matrix $\mathbf{W}$
Hyperparameters: n_c (no. of clusters), n_v (number of eigenvectors), k, σ (for constructing $\mathbf{W}$)

1. compute Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{W}$
2. normalize Laplacian $\mathbf{L}_{\mathrm{sym}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$
3. compute $\mathbf{u}_1, \dots, \mathbf{u}_{n_v}$ the first n_v eigenvectors of $\mathbf{L}_{\mathrm{sym}}$
4. define new feature matrix

$$\mathbf{U} = \begin{pmatrix} \mathbf{u}_1^\top \\ \mathbf{u}_2^\top \\ \vdots \\ \mathbf{u}_{n_c}^\top \end{pmatrix} \in \mathbb{R}^{n_v \times n}$$

5. use k-means to cluster the columns of $\mathbf{U}$ into n_v classes

Output: $\mathbf{C} \in \mathbb{R}^{n_c \times n}(\mathbf{C}_{ij} = 1$ if example j belongs to class $i)$

Graph Laplacians in Semisupervised Learning

Goal: Given $(\mathbf{y}_1, \mathbf{c}_1), (\mathbf{y}_2, \mathbf{c}_2), \dots, (\mathbf{y}_k, \mathbf{c}_k)$ obtained using

$$\mathbf{c} = f(\mathbf{y}) + \boldsymbol{\epsilon}, \quad (\epsilon \text{ is some noise})$$

estimate $f(\mathbf{y}_{k+1}), f(\mathbf{y}_{k+2}), \dots, f(\mathbf{y}_n)$. Remarks:

- in most cases it is possible to obtain a few labeled data
- can work well if the unlabeled data is similar to the labeled one and f is smooth
- typically more robust than unsupervised learning

Example : Let $n_c = 1$. Solve optimization problem for $\mathbf{f} \in \mathbb{R}^n$

$$\min_{\mathbf{f}} \frac{1}{2} \sum_{i=1}^{k} (\mathbf{f}_i - \mathbf{c}_i)^2 + \frac{\alpha}{2} \mathbf{f}^\top \mathbf{L} \mathbf{f}, \quad \alpha > 0$$

Σ: Spectral Clustering for Unsupervised Learning

- our focus: graph-based methods
- key idea: Given $\mathbf{Y}$, create weighted undirected graph. Choices:
  - distance function (defines similarity)
  - criteria for edges (control sparsity)
  - weights for graph (usually based on distance)
- spectral clustering requires first few eigenvectors
  - naive complexity n^3
  - use iterative/randomized linear algebra for large n
- semi-supervised learning
  - use graph Laplacian to obtain smooth mapping

References

[1] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *papers.nips.cc*, 2002.

[2] A. Pothen, H. D. Simon, and K.-P. Paul Liu. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Review*, 11(3):430–452, July 1990.

[3] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Aug. 2007.