Signal Processing on Graphs

P. Vandergheynst

CIS - Center for Intelligent Systems

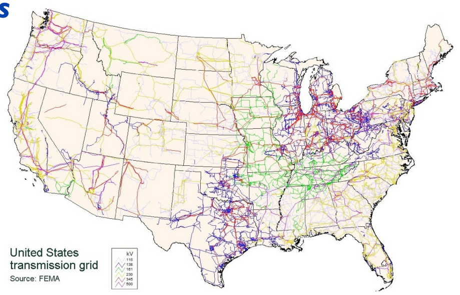
EPFL

Peyresq, June 2023

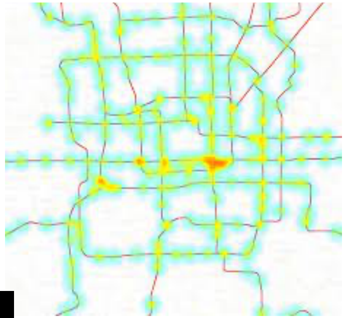
EPFL Center for Intelligent Systems

Processing Data on/with Graphs

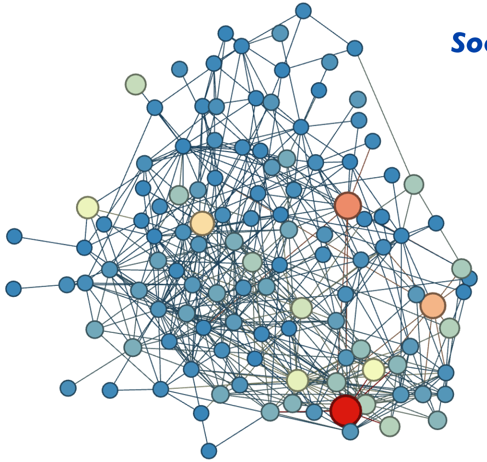
Energy Networks

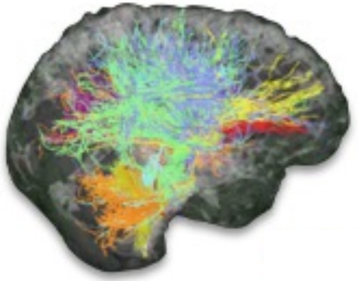


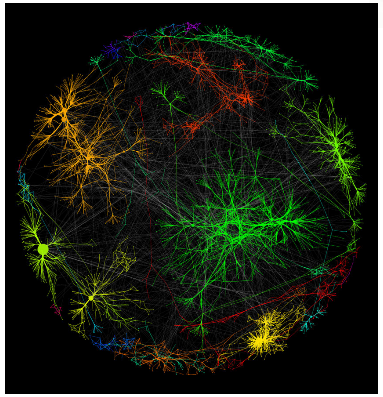
Transportation Networks



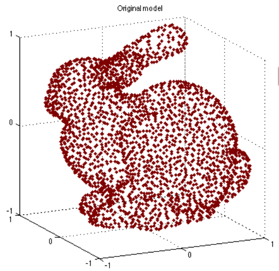
Social Networks

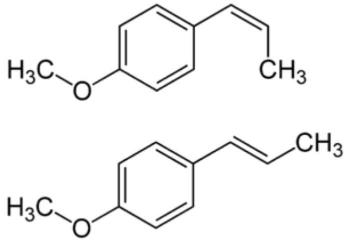


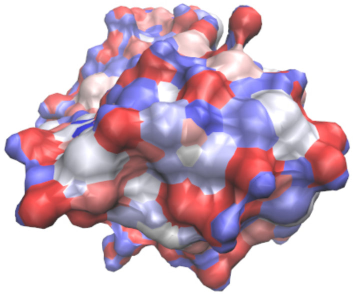




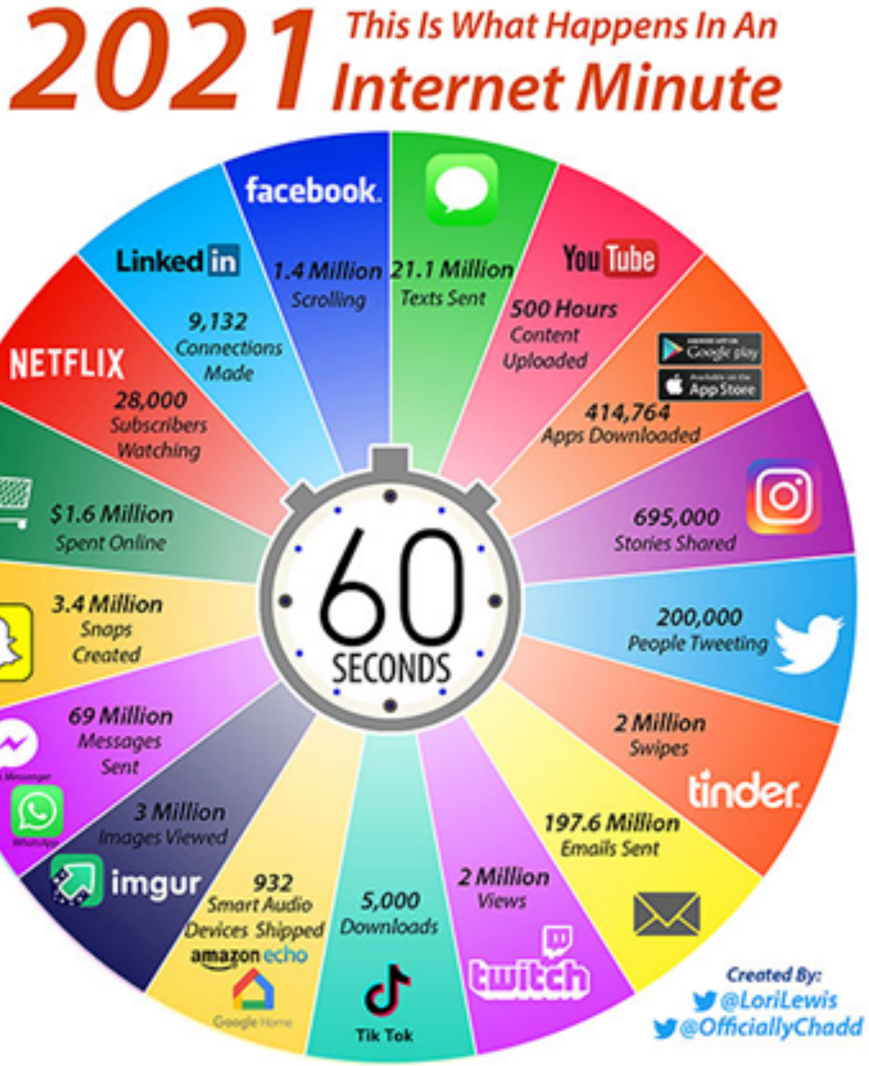
Biological Networks







Irregular Data Domains


2021 This Is What Happens In An Internet Minute
facebook 1.4 Million Scrolling
21.1 Million Texts Sent
YouTube 500 Hours Content Uploaded
Google play App Store 414,764 Apps Downloaded
695,000 Stories Shared
200,000 People Tweeting
2 Million Swipes tinder
197.6 Million Emails Sent
2 Million Views twitch
5,000 Downloads Tik Tok
932 Smart Audio Devices Shipped amazon echo Google Home
imgur 3 Million Images Viewed
69 Million Messages Sent Facebook Messenger WhatsApp
3.4 Million Snaps Created
$1.6 Million Spent Online
NETFLIX 28,000 Subscribers Watching
LinkedIn 9,132 Connections Made
60 SECONDS
Created By: @LoriLewis @OfficiallyChadd

Pattern Recognition and Machine Learning (in one slide)

Schematically Machine Learning seeks to build a map from data x to decisions y

$$y = \mathcal{F}(x)$$ the output decision can be numerical, categorical, ...

Not so long ago, this involved decomposing

$$\mathcal{F} = \boxed{\mathcal{F}_{\text{decision}}} \circ \boxed{\mathcal{F}_{\text{features}}}$$

hand-crafted using domain knowledge

learnt from data

But lately,

$$\mathcal{F} = \boxed{\mathcal{F}_{\text{decision}}} \circ \boxed{\mathcal{F}_{\text{layer } n} \circ \cdots \circ \mathcal{F}_{\text{layer } 1}}$$

learnt from data !

$$\mathcal{F} = \boxed{\mathcal{F}_{\text{decision}}} \circ \boxed{\mathcal{F}_{\text{features}}}$$

Input (attributed) Graph

$$\begin{tikzcd} & \text{Input (attributed) Graph} \arrow[dl] \arrow[dr] & \\ \text{Construct Features} & & \text{Learn Features} \end{tikzcd}$$

Construct Features

(Domain knowledge)

Spectral Clustering
Graph signal processing
Filters, wavelets

Learn Features

Decode-only, matrix factorisation

Laplacian eigenmaps, GraRep
DeepWalk, Node2Vec, LINE

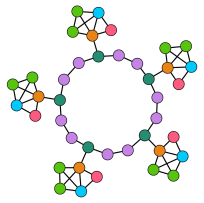
Encode-Decode

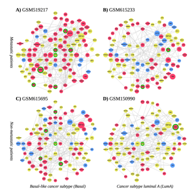
Graph Neural Nets (GNNs)
Message Passing and Beyond

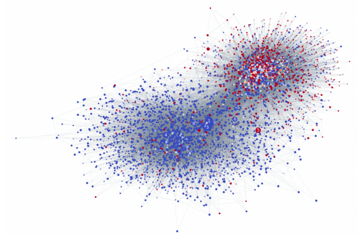
Typical Tasks

$$y = \mathcal{F}(x)$$

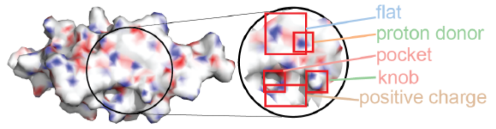
x is a node: node classification, edge prediction

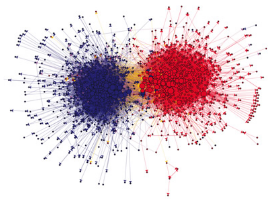




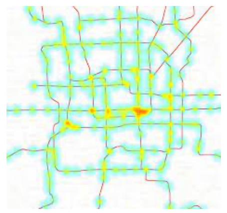


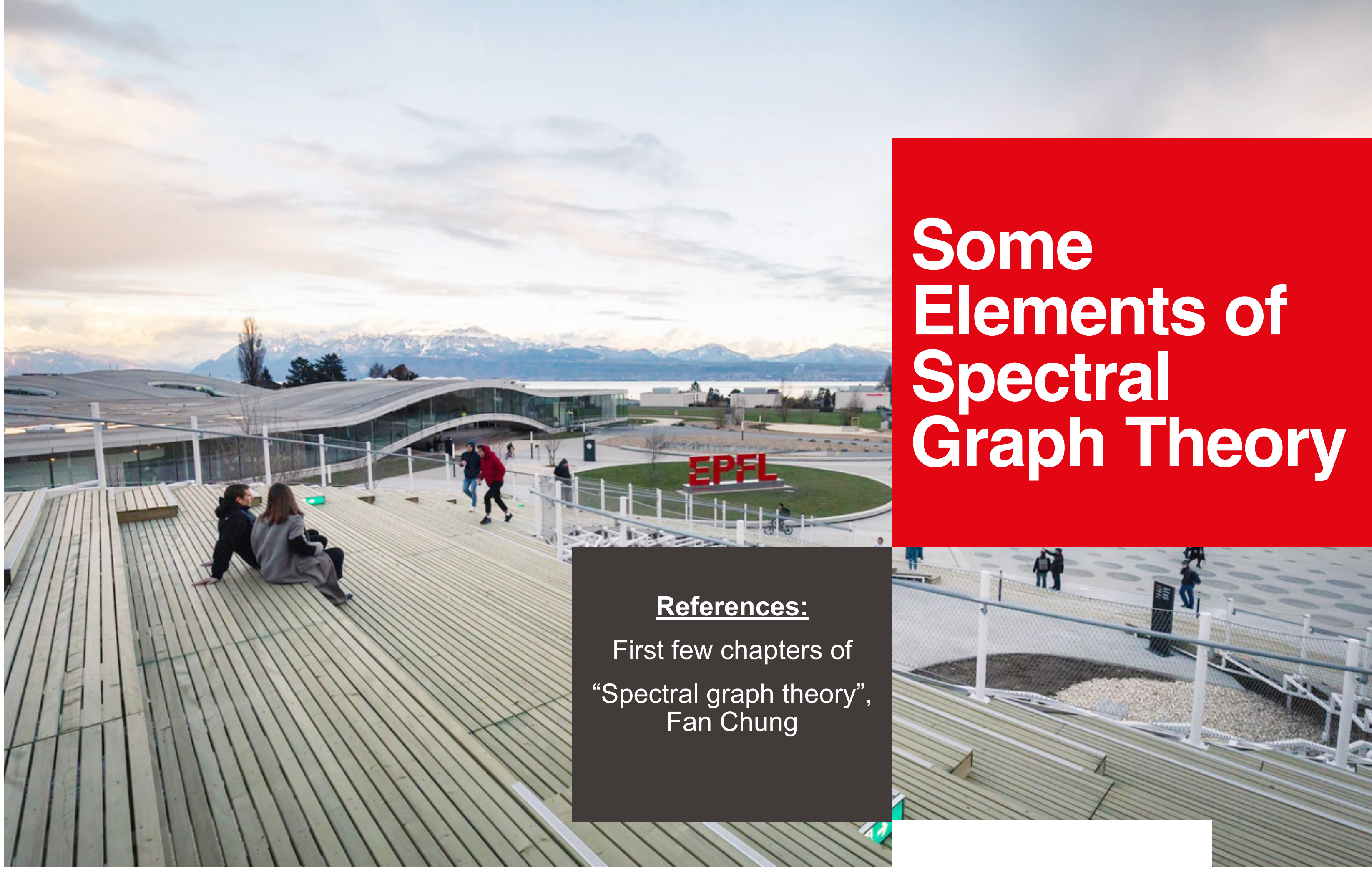
x is a (sub)graph: detecting/classifying structures or whole graphs





x is a graph signal or sample of a graph signal (node attribute): all of the above (!), classify information over a fixed network





Some Elements of Spectral Graph Theory

References:

First few chapters of

“Spectral graph theory”, Fan Chung

Orientation-agnostic definitions

$$V = \{v_1, \ldots, v_N\} \qquad E = \{e_1, \ldots, e_M\}$$

Degrees and Degree Matrix:

$$d(v) = |\{u \in V \text{s.t. } (u,v) \in E \text{ or } (v,u) \in E\}|$$

$$\mathbf{D}(G) = \operatorname{diag}(d_1, \ldots d_N)$$

Incidence Matrix:

$$\mathbf{S}(i,j) = \begin{cases} +1 & \text{if } e_j = (v_i, v_k) \text{ for some } k \\ -1 & \text{if } e_j = (v_k, v_i) \text{ for some } k \\ 0 & \text{otherwise} \end{cases}$$

Orientation-agnostic definitions

$$V = \{v_1, \ldots, v_N\} \qquad E = \{e_1, \ldots, e_M\}$$

Adjacency matrix

$$\mathbf{A}(i,j) = \begin{cases} +1 & \text{if there is an edge } (v_i, v_j) \text{ or } (v_j, v_i) \in E \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{A}(i,j) = \begin{cases} +1 & \text{if there is an edge } \{v_i, v_j\} \in E \\ 0 & \text{otherwise} \end{cases}$$

Some useful definitions

Walk: finite/infinite set of edges $(e_1, e_2, \dots)$ for which there is a sequence of vertices $(v_1, v_2, \dots)$ such that $e_i = (v_i, v_{i+1})$

Trail: a walk with all distinct edges

Path: a trail with all distinct vertices as well

Diameter: $\mathrm{diam}(G)$ Length of the longest shortest path

Volume of a subset: $\mathrm{vol}(A) = \sum_{i \in A} d(i)$

Extensions to weighted graphs

$V = \{v_1, \ldots, v_N\}$ $\qquad$ $E = \{e_1, \ldots, e_M\}$

Weight Matrix:

A *symmetric* N-by-N matrix $\mathbf{W}$

$\mathbf{W}(i,j) \geqslant 0 \qquad \mathbf{W}(i,i) = 0$

$\mathbf{W}(i,j)$ is the weight ("strength") of the edge between i,j (if any)

Degrees: $$d(v_i) = \sum_{j \sim i} \mathbf{W}(i,j)$$

Orientation-agnostic definitions

With these definitions we have:

$$\mathbf{S}\mathbf{S}^T = \mathbf{D} - \mathbf{A}$$

$\mathbf{L} = \mathbf{D}$ - $\mathbf{A}$ is called unnormalized Laplacian of G

$\mathbf{L}$ does not depend on the orientation (so OK for undirected)

For a weighted graph we have $\mathbf{L} = \mathbf{D}$ **-** $\mathbf{W}$ (attention to degrees)

$\mathbf{L}$ is a symmetric, positive semi-definite matrix

Graph Laplacian

Proposition: **L** is positive semi-definite

For any N-by-N weight matrix **W**, if $\mathbf{L} = \mathbf{D}\text{-}\mathbf{W}$ where **D** is the degree matrix of **W**, then

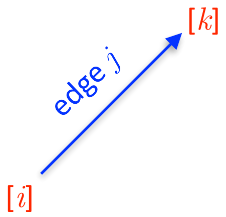
$$x^T\mathbf{L}x = \frac{1}{2}\sum_{i,j}\mathbf{W}(i,j)(x[i]-x[j])^2 \geqslant 0 \quad \forall x \in \mathbb{R}^N$$

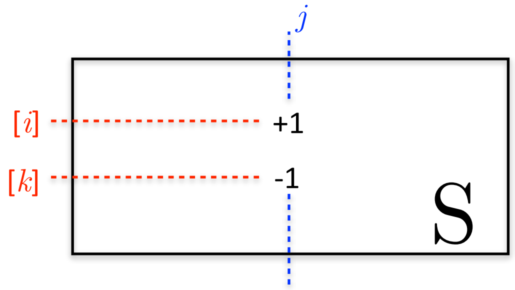
Rem: to ease notations we will sometimes use $w_{ij} = \mathbf{W}(i,j)$

x is a (component of) a node attribute

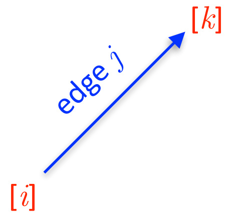
Incidence Matrix: $\mathbf{S} \in \mathbb{R}^{N \times M} \quad N = |V|, \quad M = |E|$

$$\mathbf{S}(i,j) = \begin{cases} +1 & \text{if } e_j = (v_i, v_k) \text{ for some } k \\ -1 & \text{if } e_j = (v_k, v_i) \text{ for some } k \\ 0 & \text{otherwise} \end{cases}$$





Attribute, signal (function) f defined on the vertices $f \in \mathbb{R}^N$



$(\mathbf{S}^T f)[j] = f[i] - f[k]$ derivative of f along edge j

$F = \mathbf{S}^T f \in \mathbb{R}^M$ gradient of f ($F =$ edge-based signal)

$g = \mathbf{S}G \in \mathbb{R}^N$ divergence of g ($G =$ edge-based signal)

$\mathbf{L} = \mathbf{S}\mathbf{S}^T$

$$\begin{aligned} f^T \mathbf{L} f &= f^T \mathbf{S}\mathbf{S}^T f \\ &= \|\mathbf{S}^T f\|_2^2 \\ &= \sum_{i \sim k} (f[i] - f[k])^2 \end{aligned}$$

In general for a weighted graph: $f^T \mathbf{L} f = \sum_{i \sim k} \mathbf{W}(i,k)(f[i] - f[k])^2$

This quadratic (Dirichlet) form is a measure of how smooth the signal is

Graph Laplacian

Since $\mathbf{L}$ is real, symmetric and PSD:

- It has an eigen decomposition into real eigenvalues and eigenvectors λ_i, u_i
- The eigenvalues are non-negative $0 = \lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_N$

$$\updownarrow$$

$$\mathbf{L}\mathbf{1} = 0$$

What can be learned from eigenvectors and eigenvalues ?

Some examples

Path graph

$$\begin{bmatrix} 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & & & \\ & 2 & & & & \\ & & 2 & & & \\ & & & \ddots & & \\ & & & & 2 & \\ & & & & & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 1 \\ & & & & 1 & 0 \end{bmatrix}$$

$$\lambda_k = 2 - 2\cos\frac{\pi k}{N} = 4\sin^2\frac{\pi k}{2N},\ k = 0, ..., N-1$$

$$u_k[\ell] = \cos\left(\pi k(\ell + \frac{1}{2})/N\right),\ \ell = 0, ..., N-1$$

DCT II transform

Some examples

Ring graph

$$\begin{pmatrix} 2 & -1 & & & -1 \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ -1 & & & -1 & 2 \end{pmatrix}$$

$$\lambda_k = 2 - 2\cos\frac{\pi k}{N} = 4\sin^2\frac{\pi k}{2N},\ k = 0, ..., N-1$$

$$u_k^c[\ell] = \cos\Big(2\pi k\ell/N\Big),\ \ell = 0, ..., N-1$$

DCT transform

$$u_k^s[\ell] = \sin\Big(2\pi k\ell/N\Big),\ \ell = 0, ..., N-1$$

Proposition: eigen decomposition of $\mathbf{L}$ and structure of G

The number of connected components c of G is the dimension of the null space of $\mathbf{L}$. Furthermore the null space of $\mathbf{L}$ has a basis of indicator vectors of the connected components of G

Indicator of a subset H of V is

$$x \in \mathbf{R}^N \text{ s.t. } \begin{cases} x[i] = 1 & \text{if } \in H \\ x[i] = 0 & \text{otherwise} \end{cases}$$

Normalized Graph Laplacian

Note: we will sometimes need to consider the generalised problem

$$\mathbf{L}u = \lambda \mathbf{D}u$$

In this case it makes sense to introduce the normalised Laplacian

$$\mathbf{L}_{\text{norm}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$$

Eigenvectors are closely related

$$\mathbf{L}_{\text{norm}} f = \lambda f \rightarrow u = \mathbf{D}^{-1/2} f$$

Normalized Graph Laplacian

Eigenvalues of the normalised Laplacian

$$0 = \lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_N \leqslant 2$$

Algebraic connectivity

IFF bipartite graph!

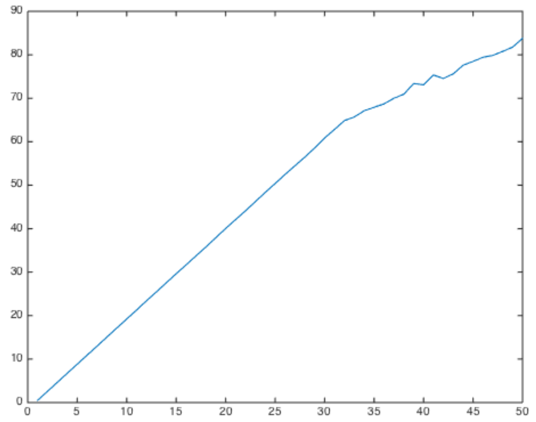
Algebraic Connectivity, Fiedler Vector

Multiplicity of eigenvalue 0 gives connectedness of graph

What if $\lambda_2 > 0$?

Experiment:

Gradually increase connections between two Erdos-Renyi subgraphs



$$\lambda_2 \geqslant \frac{1}{\mathrm{vol}(G)d(G)}$$ where $d(G)$ is the diameter of the graph

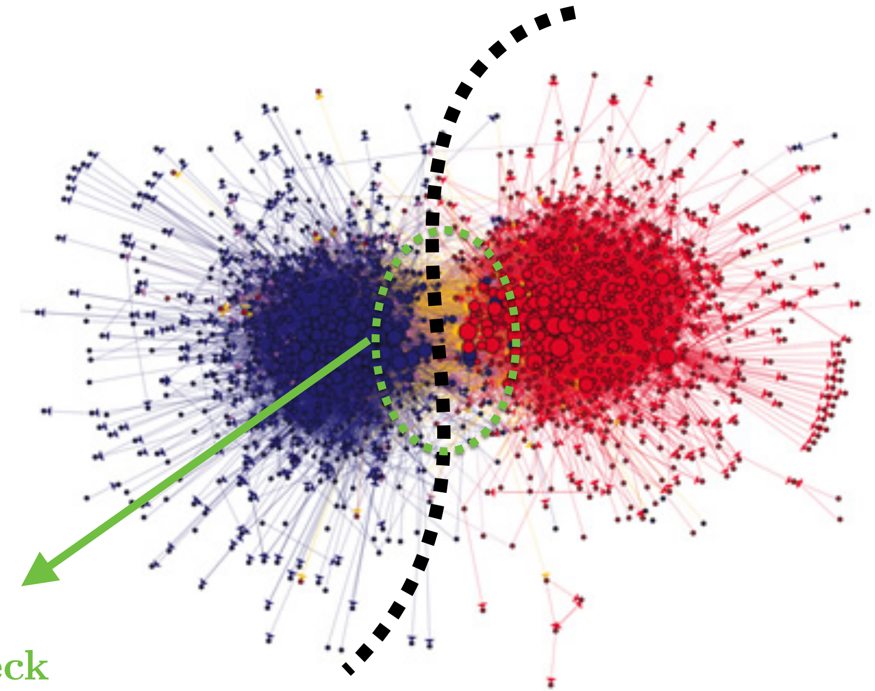
The Cheeger Constant

Cheeger constant measures presence of a "bottleneck"

$A \subset V \qquad \partial A = \left\{(u,v) \in E \text{ s.t. } u \in A,\, v \in \overline{A}\right\}$

$$h(G) = \min_{A \subset V} \left\{ \frac{|\partial A|}{\min\Big(\mathrm{vol}(A), \mathrm{vol}(\overline{A})\Big)} \text{ s.t. } 0 < |A| < \frac{1}{2}|V| \right\}$$

The Cheeger Constant



Bottleneck

A Cheeger Inequality

The Cheeger constant and algebraic connectivity are related by Cheeger inequalities. A simple example:

Theorem: Cheeger Inequality[Polya, Szego]

For a general graph G,

$$2h(G) \geqslant \lambda_2 \geqslant \frac{h^2(G)}{2}$$

Remark: the eigenvector associated to the algebraic connectivity is called the Fiedler vector

Algebraic Connectivity, Fiedler Vector

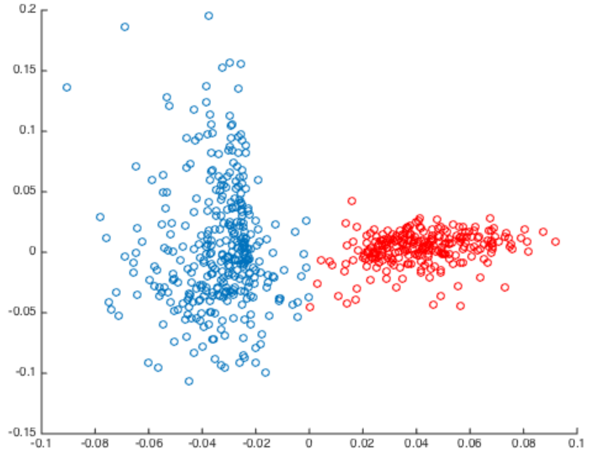
Set of 1490 US political blogs, labelled “Dem” or “Rep”

Hyperlinks among blogs

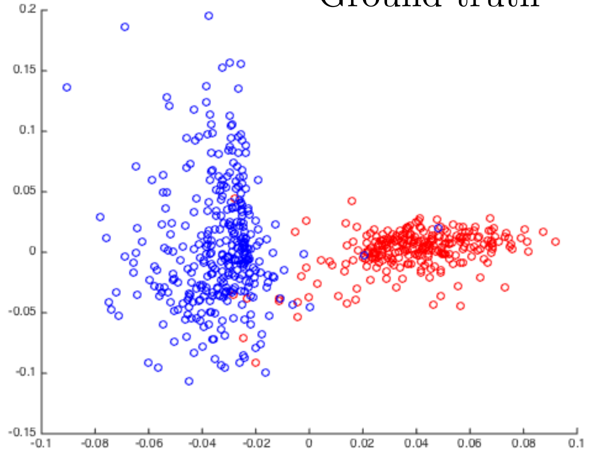
Removed small degrees (<12), keep N = 622 vertices

Compute normalised Laplacian, Fiedler vector

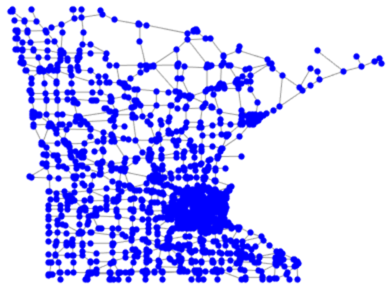
Assign attributes +1, -1 by sign of Fiedler vector

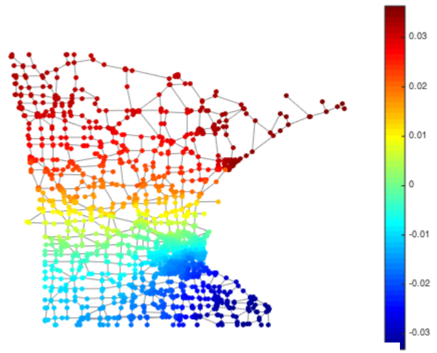


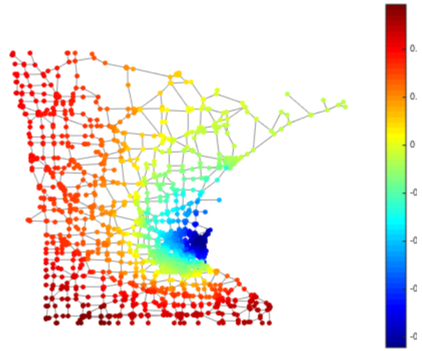
Ground truth

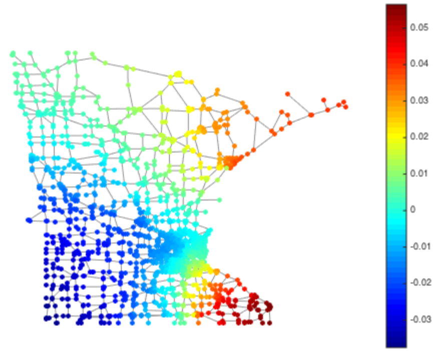


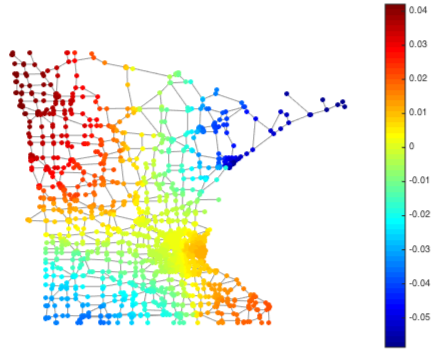
A Few Laplacian Eigenvectors











Spectral Clustering

References:

U. Von Luxburg, “A tutorial on spectral clustering,” *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007.

Back to the Start: Cut and Cluster

When cutting through edges, we can associate cost functions inspired by the Cheeger constant:

$$C(A,B) := \sum_{i \in A, j \in B} \mathbf{W}[i,j]$$

$$\text{RatioCut}(A,\overline{A}) := \frac{1}{2}\frac{C(A,\overline{A})}{|A|} + \frac{1}{2}\frac{C(A,\overline{A})}{|\overline{A}|}$$

$$\text{NormalizedCut}(A,\overline{A}) = \frac{1}{2}\frac{C(A,\overline{A})}{\text{vol}(A)} + \frac{1}{2}\frac{C(A,\overline{A})}{\text{vol}(\overline{A})}$$

Normalization seeks to impose balanced clusters

Exposing RatioCut

Let's try to solve: $\min_{A \subset V} \mathrm{RatioCut}(A, \overline{A})$

Observations: $$f[i] = \begin{cases} \sqrt{|\overline{A}|/|A|} & \text{if } i \in A \\ -\sqrt{|A|/|\overline{A}|} & \text{if } i \in \overline{A} \end{cases}$$

Sign of *f* is an indicator of the partition

$$\Longleftrightarrow \quad f^T \mathbf{L} f = |V| \mathrm{RatioCut}(A, \overline{A})$$

$$\|f\| = \sqrt{|V|} \text{ and } \langle f, 1 \rangle = 0$$

Exposing RatioCut

The following problem is equivalent to Ratiocut:

$$\arg\min_{\boxed{A\subset V}} f^T \mathbf{L} f \text{ subject to } \|f\| = \sqrt{N}, \quad \langle f, 1\rangle = 0 \text{ and } \boxed{f \text{ indicator of } A}$$

NP-hard

$$\arg\min_{f} f^T \mathbf{L} f \text{ subject to } \|f\| = \sqrt{N}, \quad \langle f, 1\rangle = 0$$

Relaxed problem: **Looking for a <u>smooth</u> partition function!**

Exposing RatioCut

$$\arg\min_f f^T \mathbf{L} f \text{ subject to } \|f\| = \sqrt{N}, \quad \langle f, 1 \rangle = 0$$

Solution (G connected): eigenvector of λ_2

Warning: recover partition after thresholding $f = \mathrm{sign}(u_2)$

So we are back to the Fiedler vector !!!

RatioCut: Generalizing to $k > 2$

For more than two components, we look for a set of partition functions

$$F \in \mathbb{R}^{N \times k} \qquad F[i,j] = f_j[i] = \begin{cases} 1/\sqrt{|A_j|} & \text{if } v_i \in A_j \\ 0 & \text{otherwise} \end{cases}$$

At each vertex we look for binary assignment vectors

Observe: $$f_j^T \mathbf{L} f_j = \frac{\mathrm{Cut}(A_j, \overline{A_j})}{|A_j|} \qquad F^T F = \mathbb{I}$$

$$\mathrm{RatioCut}(A_1, \ldots, A_k) = \mathrm{Tr}(F^T \mathbf{L} F)$$

Suggests the relaxed problem:

$$\arg \min_{F \in \mathbb{R}^{N \times k}} \mathrm{Tr}(F^T \mathbf{L} F) \text{ such that } F^T F = \mathbb{I}$$

Unnormalized Spectral Clustering

This form of relaxed RatioCut = Unnormalized Spectral Clustering

$$\arg \min_{F \in \mathbb{R}^{N \times k}} \mathrm{Tr}(F^T \mathbf{L} F) \text{ such that } F^T F = \mathbb{I}$$

At each vertex we look for feature vectors

Algorithm: Unnormalized Spectral Clustering

Compute the matrix F of first k eigenvectors of L

Apply k-means to rows of F to obtain cluster assignments

Normalized Cut, k=2

$$\text{NormalizedCut}(A,\overline{A}) = \frac{1}{2}\frac{C(A,\overline{A})}{\text{vol}(A)} + \frac{1}{2}\frac{C(A,\overline{A})}{\text{vol}(\overline{A})}$$

$$f[i] = \begin{cases} \sqrt{\text{vol}(\overline{A})/\text{vol}(A)} & \text{if } v_i \in A \\ -\sqrt{\text{vol}(A)/\text{vol}(\overline{A})} & \text{otherwise} \end{cases}$$

Check that: $\langle \mathbf{D}f, 1\rangle = 0 \qquad f^T\mathbf{D}f = \text{vol}(G)$

$$f^T\mathbf{L}f = \text{vol}(V)\text{NormalizedCut}(A,\overline{A})$$

$$\arg\min_f f^T\mathbf{L}f \text{ subject to } f^T\mathbf{D}f = \text{vol}(G), \quad \langle \mathbf{D}f, 1\rangle = 0$$

$$\Big\Updownarrow \quad g = \mathbf{D}^{1/2}f$$

$$\arg\min_g g^T\mathbf{L}_{\text{norm}}g \text{ subject to } \|g\|^2 = \text{vol}(G), \quad \langle g, \mathbf{D}^{1/2}1\rangle = 0$$

Normalized Cut, k> 2

$$F[i,j] = f_j[i] = \begin{cases} 1/\sqrt{\text{vol}(A_j)} & \text{if } v_i \in A_j \\ 0 & \text{otherwise} \end{cases}$$

$$f_j^T \mathbf{L} f_j = \frac{\text{Cut}(A_j, \overline{A_j})}{\text{vol}(A_j)} \qquad F^T F = \mathbb{I} \qquad f_j^T \mathbf{D} f_j = 1$$

$$\arg \min_{H \in \mathbb{R}^{N \times k}} \text{Tr}(H^T \mathbf{L}_{\text{norm}} H) \text{ such that } H^T H = \mathbb{I} \qquad H = \mathbf{D}^{1/2} F$$

Algorithm: Normalized Spectral Clustering

Compute the matrix H of first k eigenvectors of $\mathbf{L}_{\text{norm}}$

Apply k-means to rows of H to obtain cluster assignments

Applications

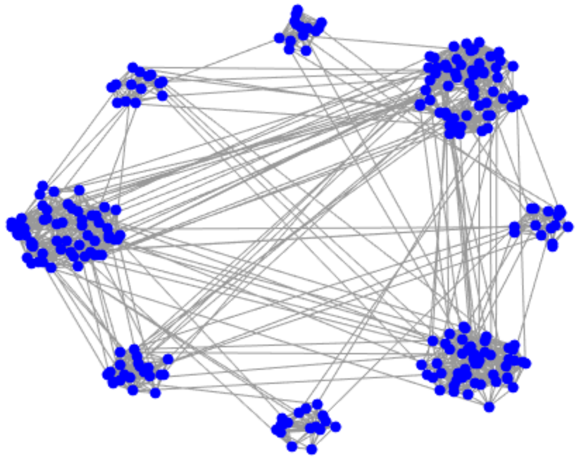
In practice normalised spectral clustering is often preferred

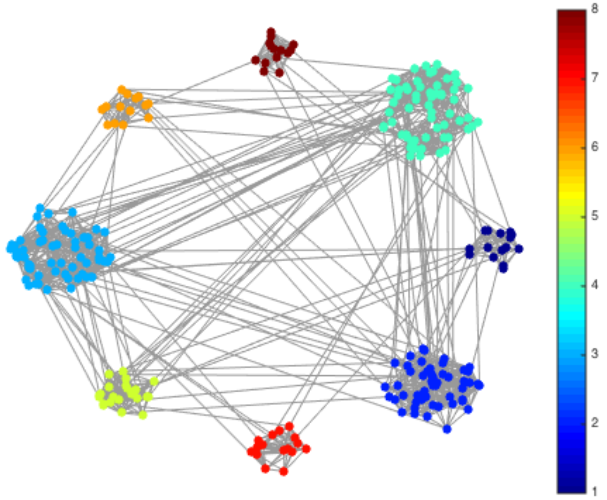
In practice the eigenvectors are "re-normalized" by the degrees $F = \mathbf{D}^{-1/2} H$ before k-means, because these are real cluster assignments

Rem: this is equivalent to using the "random walk Laplacian"

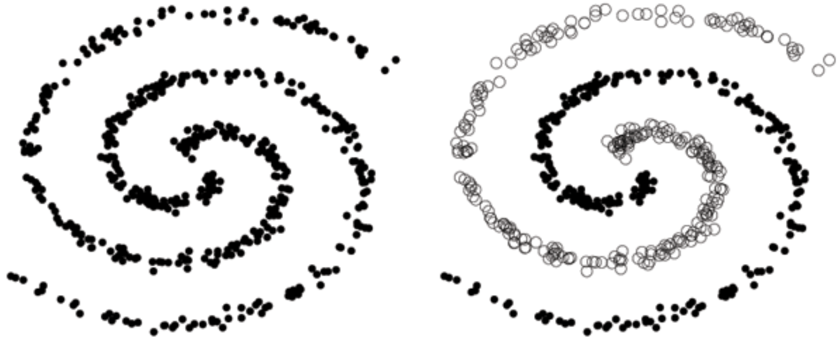
$$\mathbf{L}_{\mathrm{rw}} = \mathbf{D}^{-1}\mathbf{L}$$

If data has k clear clusters, there will be a gap in the Laplacian spectrum after the k-th eigenvalue. Use to choose k.



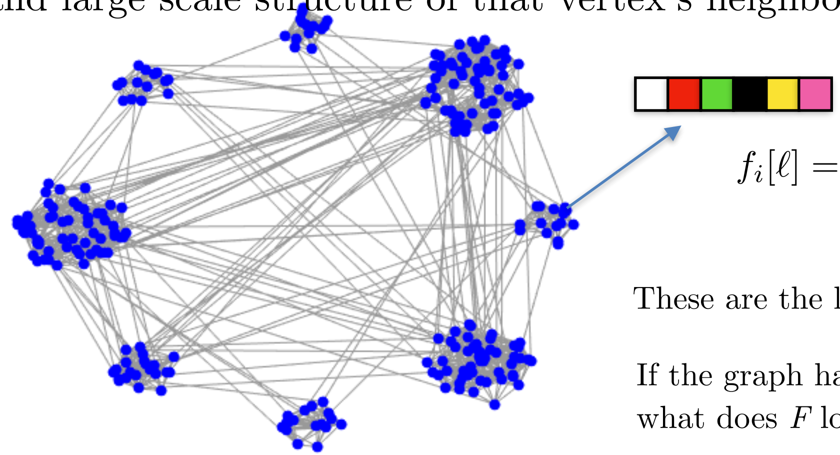

8
7
6
5
4
3
2
1

Example



What is the algorithm doing - view 1

At each vertex the algorithm associates a feature vector that represents the fine and large scale structure of that vertex's neighbourhood in the graph



vertex $[i]$, feature $f_i \in \mathbb{R}^k$

$$f_i[\ell] = u_\ell[i], \quad \ell = 1, \ldots, k$$

These are the lines of the matrix F

If the graph has k disconnected components, what does F look like ?

k-means is then applied to these vectors to cluster into k clusters

In short, we transform the graph into a feature matrix and partition it.
These features are selected because they discriminate our classes. Classic unsupervised ML.

What is the algorithm doing - view 2

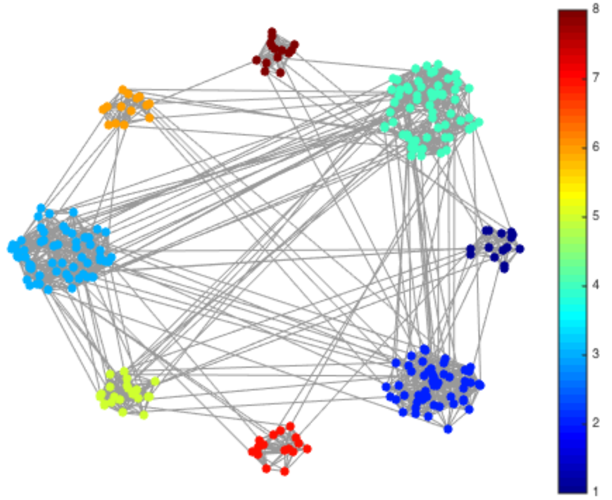
We are looking for k "partition signals (functions)"

$$f_\ell : V \mapsto \mathbb{R}$$

In the ideal case (k disconnected components)

$$f_\ell[i] = \begin{cases} c_i & \text{if } i \in \text{cluster } \ell \\ 0 & \text{otherwise} \end{cases}$$

These are maximally smooth graph signals: $f_\ell^T \mathbf{L} f_\ell = 0$



Laplacian Eigenmaps

References:

Belkin, M. & Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. Neural Computation 15, 1373–1396 (2003).

Spectral Graph Embedding

Dataset is a large matrix $X \in \mathbb{R}^{N \times L}$ N is the number of data points

L is the dimension of each data points

Often $L >> 1$ and must be reduced (think images)

For computations

For visualisation, in which case we would like $L = 2, 3$

Q: can we learn a low-dimensional embedding (a latent vector for each data point) that preserves the original structure of X ?

Formulation

Find a mapping from the N high-dim data points to N low-dim points

$$x_1, \ldots, x_N \mapsto y_1, \ldots, y_N$$

$$x_i \in \mathbb{R}^L \qquad y_i \in \mathbb{R}^P$$

Assumption: we have a graph of similarities among original data points

Similarities are often constructed by either :

selecting k-nearest neighbours of each point with distance $d(x_i, x_j)$

OR

selecting all points in a neighbourhood $d(x_i, x_j) \leqslant \epsilon$

THEN

weighting these edges ex: $\mathbf{W}(i,j) = e^{-d(x_i,x_j)^2/t}$

Formulation

W captures similarities among data points $x_i \in \mathbb{R}^L$

Suppose we embed in 1 dimension (P=1)

$$\arg \min_{y_1,\ldots,y_N} \sum_{i \sim j} \mathbf{W}(i,j)(y_i - y_j)^2 \Rightarrow \arg \min_{y \in \mathbb{R}^N} y^T \mathbf{L} y$$

Add a constraint to avoid collapse y=0: $y^T \mathbf{D} y = 1$

Avoid trivial eigenvector: $y^T \mathbf{D} \mathbf{1} = 0$

$$\Rightarrow \arg \min_{\substack{y \in \mathbb{R}^N \\ y^T \mathbf{D} y = 1 \\ y^T \mathbf{D} \mathbf{1} = 0}} y^T \mathbf{L} y$$

Full problem

When we embed in P dimension $(P > 1)$

$$\arg\min_{y_1,\ldots,y_N} \sum_{i\sim j} \mathbf{W}(i,j)\|y_i - y_j\|_2^2$$

Algorithm: Laplacian Eigenmaps

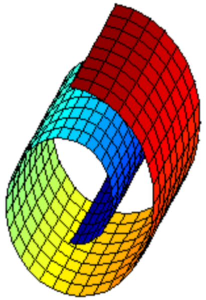
Collect the coordinates of embedded points as lines of matrix Y

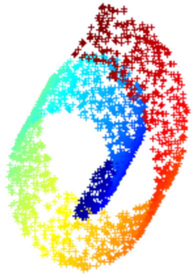
$$\arg \min_{\substack{Y \in \mathbb{R}^{N\times P} \\ Y^T\mathbf{D}Y = \mathbb{I}}} \operatorname{tr}(Y^T\mathbf{L}Y)$$

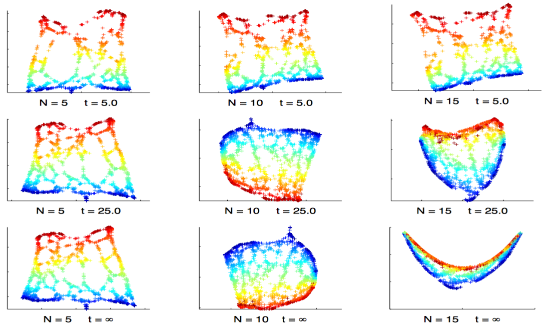
Laplacian Eigenmaps produces coordinate maps that are smooth functions/signals over the original graph.

Note similarity with clustering !

Examples: synthetic

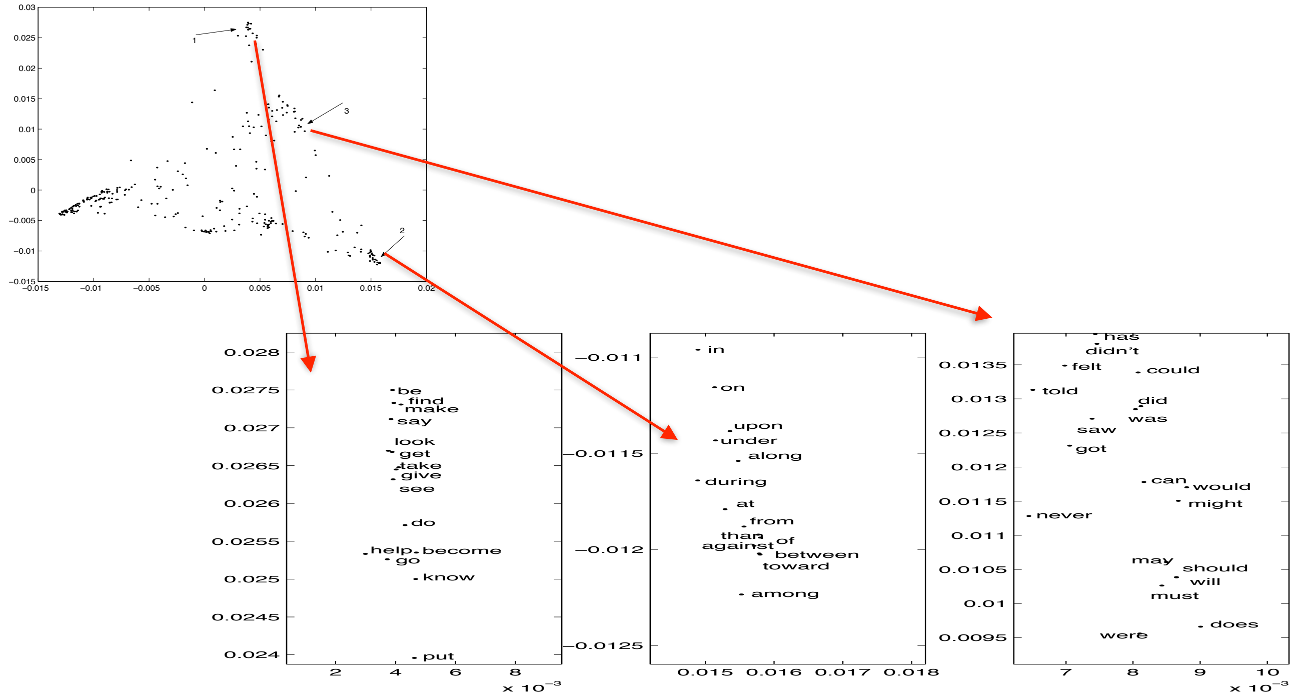






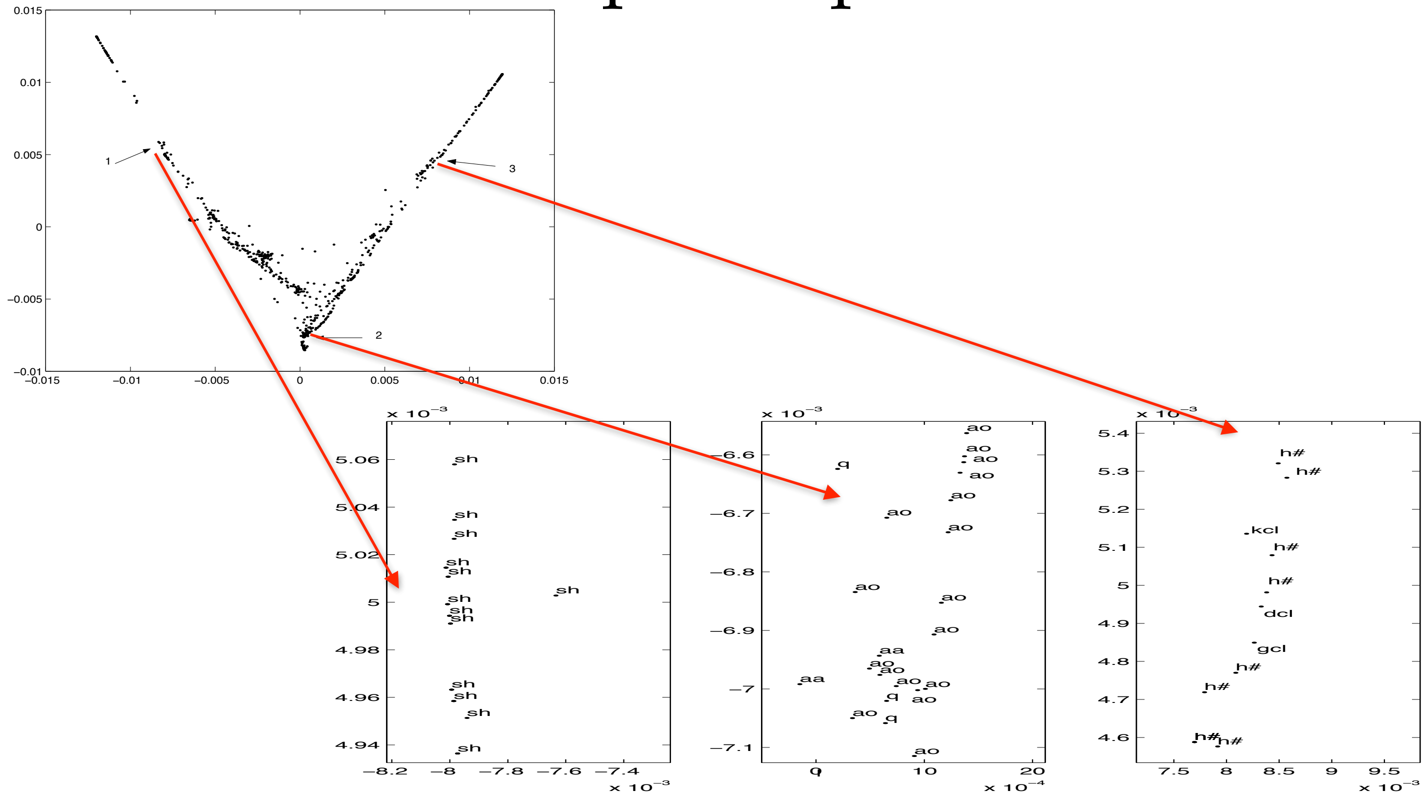
M. Belkin and P. Niyogi, “Laplacian eigenmaps for dimensionality reduction and data representation,” *Neural Comput*, vol. 15, no. 6, pp. 1373–1396, 2003.

Examples: text



M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Comput*, vol. 15, no. 6, pp. 1373–1396, 2003.

Examples: speech



Other techniques for low-dimensional graph embedding

References:

Hamilton, W. L., Ying, R. & Leskovec, J. Representation Learning on Graphs: Methods and Applications. Arxiv (2017).

Inner-product based factorisation

Simple idea: reconstruct a measure of proximity (adjacency or more complex) using an inner product decoder of embedding (latent) variables

Roughly, if $Z \in \mathbb{R}^{d \times n}$ contains the embedding $z_i \in \mathbb{R}^d$ of nodes we're looking to solve the following problem:

$$Z^* = \arg \min_{Z \in \mathbb{R}^{d \times n}} \|Z^T Z - \boxed{S}\|_F^2$$

$$= \sum_{(i,j)} \left(\boxed{z_i^T z_j} - \boxed{S_{ij}} \right)^2$$

proximity measure (ex: S = A)

inner-product decoder

Inner-product based factorisation

Two examples

1. Distributed large scale natural graph factorisation ($S = A$):

$$\begin{aligned} Z^* &= \arg \min_{Z \in \mathbb{R}^{d \times n}} \|Z^T Z - S\|_F^2 + \frac{\lambda}{2}\|Z\|_F^2 \\ &= \sum_{(i,j)} \left(z_i^T z_j - S_{ij}\right)^2 + \frac{\lambda}{2}\sum_i \|z_i\|_2^2 \end{aligned}$$

solved with SGD in p with vertex partitioning for large graphs

Inner-product based factorisation

Two examples

1. Distributed large scale natural graph factorisation ($S = A$)
2. GraRep: models k-hops relationships ($S = D^{-1}A$) $\quad p_k(x_i|x_j) = S_{ij}^k$

Inner product decoder: $\sigma(w_j^T c_i) \approx p_k(x_i|x_j)$ target and context latent vectors

k-hop Cross entropy loss: $L_k = \sum_{j \in V} L_k(w_j)$

$$L_k(w_j) = \sum_{i \in V} p_k(x_i|x_j) \log \sigma(w_j^T c_i) + \lambda \mathbb{E}_{c' \sim p_k(V)} \{\log \sigma(-w_j^T c')\}$$

noise contrastive sampling: choose c' from a noise distribution (here: at random independently of target w) and maximise probability that it is **not** a context of w

Inner-product based factorisation

Two examples

1. Distributed large scale natural graph factorisation ($S = A$)
2. GraRep: models k-hops relationships ($S = D^{-1}A$) $p_k(x_i|x_j) = S_{ij}^k$

Inner product decoder: $\sigma(w_j^T c_i) \approx p_k(x_i|x_j)$ target and context latent vectors

Choice of negative sampling distribution allows a factorization-based solution for the product $W_{(k)}^T C_{(k)} = Y_{(k)}$

$$Y_{(k)ij} = \log\left(\frac{A_{ij}^k}{\sum_{\ell \in V} A_{\ell,j}^k}\right)$$

Solve for $W_{(k)}$ by SVD

and concatenate k=1,…,K

$$Y_{(k)} \approx U_{(k)}^{(d)} \Sigma_{(k)}^{(d)} \left(V_{(k)}^{(d)}\right)^T$$

Inner-product based factorisation

Two examples

1. Distributed large scale natural graph factorisation ($S = A$)
2. GraRep: models k-hops relationships ($S = D^{-1}A$) $p_k(x_i|x_j) = S^k_{ij}$
3. DeepWalk or Node2Vec use random walks as proxy for structure

Limitations

These techniques are <u>transductive</u>: you learn embeddings of observed nodes but you don't obtain a way to <u>directly</u> compute embeddings to unseen nodes. They don't easily leverage node features. No parameter sharing among nodes.

What have we learned ?

Structure of a graph neatly summarised by powers of its Laplacian

Data matrices can be turned into graphs, hypothesising that homophily is meaningful

Spectral analysis of the Laplacian provides background for structure and shortcut for homophily via smoothness of “graph signals” (other structural proxies can be used as well)

The lack of an encoder - a direct way to map a single (attributed) node to its latent code - is a weakness

Let’s dive into graph signals and ways to manipulate them in Part II