Lecture 13 - Spectral Clustering

Monday, March 14, 2022 8:04 AM

Quick review:

Community detection (CD)

→ Finding dense regions in a network

→ Define "dense"

* cut edges / internal edges
* modularity
* conductance

Approaches

→ agglomerative, divisive heuristics

→ optimize: modularity
edge cut ↑

Evaluation

→ relative to metrics

→ relative to "ground truth"

Today: new approach

local clusterings

today: new - to

→ spectral clusterings

Spectral graph theory (/mining)

Basically: study and applications of eigenvalues/vectors of a graph's adjacency matrix or a modified adjacency matrix

PageRank:

- spectral algo. on transition probability matrix
- $M = (D^{-1}A)^T \rightarrow Mx = \lambda x$

  $Mx = x$ as $\lambda = 1$
- Note: we also talked about eigenvector centrality

  $Ax = \lambda x$

Other applications:

- Clustering

- Clustering
- Partitioning
- Coarsening
- Visualization

Graph Laplacian

Graph Laplacian L

→ $L = D - A$

→ D = diagonal degree matrix

Normalized Laplacian

$L_N = I - D^{1/2} A D^{1/2}$

↑ identity matrix

↳ fixes largest eigenvalue to be 1

(like trans. fxn prob. matrix)

Overall: mapping discrete → continuous space

Spectral clustering

spectral decomposition

$L = U \Lambda U^T$

Λ = diagonal matrix of eigenvalues

U = associated eigenvectors

↳ since L is symmetric all of Λ are real, positive, and non-increasing

Fiedler vector/value

→ first non-null eigenvalue of our Laplacian /vector.

Fiedler value is related to the connectivity of G

larger = more connected

smaller = less connected

Bi-clustering algorithm

→ we compute Fiedler vector

→ we compute Fiedler vector

→ we get values for each vertex

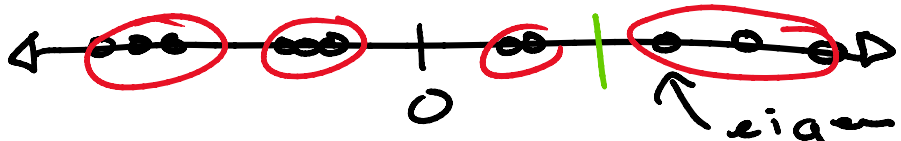
→ if vertex's value ≥ 0 → cluster A

value < 0 → cluster B

Intuitively: we're mapping from discrete → continuous space

For Fiedler → along a 1-D number line

Our biclustering is defined as some threshold along that line



More generally, to determine k clusters

→ we can consider clustering each vertex's eigenvector values for some k in some n-dimensional space

→ OR: recursive bisection

↓

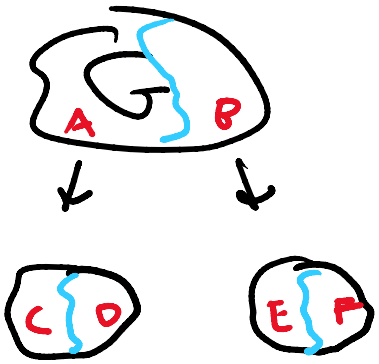
↓

we get $G = A + B$

via Fiedler cut

can then get $A = C + D$ ← subgraph of cluster

$B = E + F$



Using k-means to find k-clusters

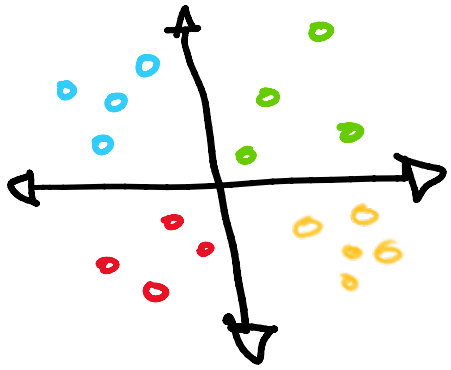
→ Often: k eigenvectors of the largest k eigenvalues

→ Alternatively: we can use $(\log_2 k)+1$ with geometric bisection

↓

we saw with 1-dimensional space for $k=2$

↓



Using n-eigenvectors to cluster into k-clusters:

→ we first compute n eigenvalues/vectors

→ each vertex gets n eigenvector values

→ these map each vertex to this n-dimensional continuous space

→ we use these coordinates for k-means clustering

→ output of k-means is our cluster assignments

k-means:

Unsupervised clustering algorithm to find k clusters in n-dimensional Euclidean space

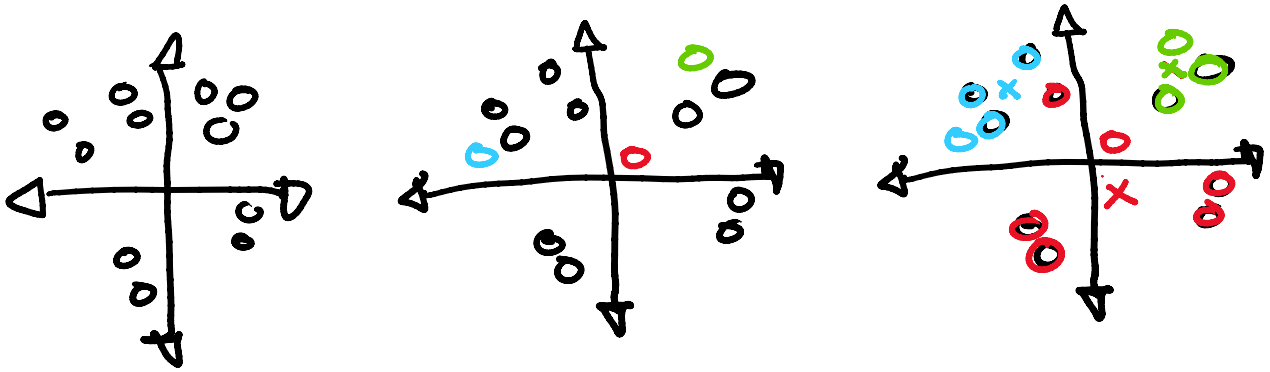
→ we initialize k points with n

→ we initialize k points with n associated coordinates

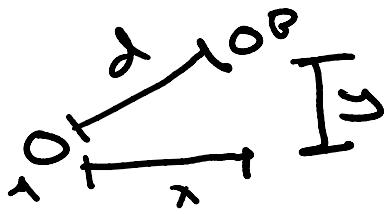
→ each vertex joins the cluster defined by smallest distance to one of those k points

→ each of k ponts updates their coordinates to the average of all members of its cluster

→ repeat until convergence



k = 3



Euclidean distance

$$d = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

Spectral partitioning

Basic graph partitioning problem:

- Given G, we want k "equal sized" parts
- Usually formulated as optimization problem:
  * minimize cut (inter-part edges)
  * constrain part sizes such that $|S_i| \approx |S_j| \; \forall i,j$

$$|S_i| \leq \frac{|V(G)|}{k} + \epsilon$$

ϵ ↑ imbalance tolerance

Note: many possible variations to this optimization problem

Why: often used in scientific computing and parallel processing

Meshes == graphs, which are what 99%+ of scientific problems use

Edge cut = amount of communication

Imbalance = workload imbalance

Imbalance = workload imbalance

Spectral partitioning

→ like clustering, but we constrain cluster sizes

→ Recall: "good" clusters have high density and therefore a low cut

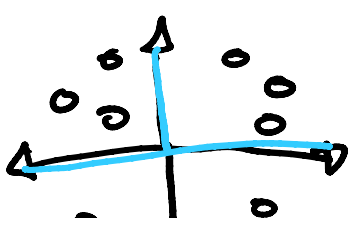
Intuitively: we can optimize cut by finding good clusters

For partitioning: these clusters should be close to equal sized

→ How: for power-2 partitioning we can use a recursive-bisection-based approach

For non-power-2: k-means but we enforce balanced cluster sizes

OR: a geometric-based approach



← we determine geometric "blocks"