L11 -- spectral clustering
[Jeff Phillips - Utah - Data Mining]

Graph G = (E,V)
  V = vertices {a,b,c,d,e,f,g,h}
  E = edges   {(a,b), (a,c), (a,d), (b,d), (c,d), (c,e), (e,f), (e,g), (f,g), (f,h)}
      unordered pairs

Draw graph:

	a	b	c	d	e	f	g	h
a	0	1	1	1	0	0	0	0
b	1	0	0	1	0	0	0	0
c	1	0	0	1	1	0	0	0
d	1	1	1	0	0	0	0	0
e	0	0	1	0	0	1	1	0
f	0	0	0	0	1	0	1	1
g	0	0	0	0	1	1	0	0
h	0	0	0	0	0	1	0	0

adjacency matrix

What are the best 2 clusters of vertices?

Top-Down Clustering:
 - find best cut into 2 (or more pieces)
 - recur on pieces

Today we'll mainly talk about finding the one best subset
  S subset V

Vol(S) = # edges with at least one edge in V
Cut(S,T) = # edges with one edge in S, and one in T

normalized cut NCut(S,T) = Cut(S,T)/Vol(S) + Cut(S,T)/Vol(T)

goal is to find cut with smallest "normalized cut" (S subset P, T = P \ S)
  - other similar measures that are also good.
  - this one gives small edges split + good balance

S = {h} -> NCut = 1/1 + 1/11 = 1.09
S = {e,f,g,h} -> NCut = 2/6 + 2/7 = 0.62

Graph as Matrix:

adjacency matrix:

A =

	a	b	c	d	e	f	g	h
a	0	1	1	1	0	0	0	0
b	1	0	0	1	0	0	0	0
c	1	0	0	1	1	0	0	0
d	1	1	1	0	0	0	0	0
e	0	0	1	0	0	1	1	0
f	0	0	0	0	1	0	1	1
g	0	0	0	0	1	1	0	0
h	0	0	0	0	0	1	0	0

degree matrix: "diagonal matrix"

D =

	a	b	c	d	e	f	g	h
a	3	0	0	0	0	0	0	0
b	0	2	0	0	0	0	0	0
c	0	0	3	0	0	0	0	0
d	0	0	0	3	0	0	0	0
e	0	0	0	0	3	0	0	0
f	0	0	0	0	0	3	0	0
g	0	0	0	0	0	0	2	0
h	0	0	0	0	0	0	0	1

Laplacian matrix:

L = D - A =

	a	b	c	d	e	f	g	h
a	3	-1	-1	-1	0	0	0	0
b	-1	2	0	-1	0	0	0	0
c	-1	0	3	-1	-1	0	0	0
d	-1	-1	-1	3	0	0	0	0
e	0	0	-1	0	3	-1	-1	0
f	0	0	0	0	-1	3	-1	-1
g	0	0	0	0	-1	-1	2	0
h	0	0	0	0	0	-1	0	1

Note that each row and column sums up to 0:
 - think of D as being flow into a vertex
 - and A as the flow out of the vertex

(We'll see other useful concepts like this)

An *eigenvector* of a matrix M, is a vector v s.t.

  Mv = lambda*v,

where lambda is a scalar. lambda is the corresponding "eigenvalue."

usually restrict that ||x|| = 1

There are several eigenvectors of L (Laplacian): sort by lambda

lambda	0	.278	1.11	2.31	3.46	4	4.82
v	1/sqrt(8)	-.36	0.08	0.10	0.28	0.25	1/sqrt(2)
	1/sqrt(8)	-.42	0.18	-.64	-.38	0.25	0
	1/sqrt(8)	-.20	-.11	0.61	0.03	-.25	0
	1/sqrt(8)	-.36	0.08	0.10	0.28	0.25	-1/sqrt(2)
	1/sqrt(8)	0.17	-.37	0.21	-.54	-.25	0
	1/sqrt(8)	0.36	-.08	-.10	-.28	0.75	0
	1/sqrt(8)	0.31	-.51	-.36	0.56	-.25	0
	1/sqrt(8)	0.50	0.73	0.08	0.11	-.25	0

[V,lambda] = eig(L)   in MATLAB or OCTAVE

** Smallest eigenvalue of L (any laplacian) is 0.

** Second Smallest eigenvalue/vector is VERY important.
- it tells us how to cut the graph
- it tells us how "best" to put all vertices on a single line
+ in first eigenvector v_2, those < 0 in S, those > 0 in T
  S = {a,b,c,d}   T = {e,f,g,h}
+ can check all cuts by v_2, use one with best NCut

** Third eigenvector v_3 can be used for 4-way cut
++ above 0 v_2, above 0 v_3   S = {h}
+- above 0 v_2, below 0 v_3   T = {e,f,g}
-+ below 0 v_2, above 0 v_3   U = {a,b,d}
-- below 0 v_2, above 0 v_3   R = {c}

Tells us how to draw a graph:
  x-axis values along v_2
  y-axis values along v_3
 (scale values by 1/sqrt{lambda_i})

Or: use first k eigenvectors to embed in R^k. Then run
- k-means, or
- other Euclidean clustering algorithms.

** The smaller the eigenvalue, the more important the vector.

** Adjacency matrix does not need to be 0-1. Can fill with similarity value.
- But good to cut off small values at 0, so matrix is "sparse" makes more efficient.