Normalized Cut

We're now going to consider the problem of taking a 2D image and dividing it into meaningful groups, so that neighboring, similar pixels are grouped together. Similarity can be judged in a lot of ways, but as an example, you can think of it being based on similarity of intensity or color, or distance between pixels.

Many early approaches to this problem performed some kind of region merging or splitting. For example, we could begin with every pixel as a region, and then merge together the most similar neighboring regions, continuing this until the number of regions is small. This is a greedy approach, with the pitfalls of greedy methods (it could prematurely make a bad decision) and their advantages (speed, which was especially important when computers were slow). Normalized cut tries to define a reasonable objective function for a good segmentation, and then to approximate the solution to this.

To keep things simple, we'll just consider the problem of dividing the image into two regions, A and B. We consider the image as a graph, in which each pixel is a vertex, and the edge between two vertices represents the extent to which they seem to belong together. Edges have high weight when the pixels are nearby and similar. Let V be the set of all vertices, which we want to divide into groups A and B. We want to choose A and B so we minimize the association between them, which we call:

cut(A,B).

We can just minimize this using a min-cut algorithm, but this tends to make A or B very small. To avoid this, we normalize the cost of the cut relative to the cost of all vertices emerging from each region, so a cut is good when the connection to the rest of the image is weak relative to the connections within the region. This gives us:

Min Ncut(A,B) = cut(A,B)/assoc(A,V) + cut(B,A)/assoc(B,V)

Solving this problem turns out to be NP-hard. So we take the following strategy. We do a lot of manipulation to get this minimization in a nice form, and then relax the problem into a continuous form, in which each pixel is partly assigned to each region. This continuous problem can then be solved exactly.

W is a matrix that contains all edge weights. W(i,j) is the affinity between pixel i and j.
x is a vector indicating the region that each pixel belongs to. x_i = 1 means region A, x_i = -1 means region B.
d is a vector of the total weight of edges leaving each pixel. So d_i is the sum of all edges leaving pixel i.

$$d_i = \sum_j W(i,j)$$

D is a diagonal matrix with the elements of d on the diagonal.

I will spare you a lot of algebra (see, eg., Shi and Malik), and state that we can show that minimizing NCUT is equivalent to minimizing the following expression:

$$\frac{y^T(D-W)y}{y^T Dy}$$

where y is a variable that encodes the same information as x in a slightly different form. Define:

$$b = \frac{\sum_{x_i>0} d_i}{\sum_{x_i<0} d_i}$$

That is, b is just a constant (for a fixed assignment of pixels to regions) that gives the ratio of the "size" of regions, measured as the number of edges leaving them. Then
$y = (1+x) - b(1-x)$
So, whereas x was 1 or -1 to indicate its region, y is 2 or -2b.

As we said before, this problem is NP-hard if we restrict the values of y to be discrete, either 2 or -2b. So the strategy is to allow y to be continuous. Then our equation becomes a generalized Rayleigh quotient which has a standard solution as an eigenvalue problem.

We'll go as far as to convert the problem to a Rayleigh quotient problem. First, let:
$$z = D^{\frac{1}{2}}y$$

Then our problem becomes one of minimizing the ratio:
$$\frac{z^T D^{-\frac{1}{2}}(D-W)D^{-\frac{1}{2}}z}{z^T z}$$

This is a Rayleigh quotient problem, and is minimized by z that are the smallest eigenvectors, satisfying:
$$D^{-\frac{1}{2}}(D-W)D^{-\frac{1}{2}}z = \lambda z$$

For this z, the ratio will be the eigenvalue λ.

(To prove that this is the right solution, abbreviate:
$$M = D^{-\frac{1}{2}}(D-W)D^{-\frac{1}{2}}$$
We can diagonalize M using the orthonormal matrix, Q, so that:
$$C = Q^T MQ$$
Where C is a diagonal matrix containing the eigenvalues of M. Then, with the substitution, v = Qz, we need to minimize:

$$\frac{(Qv)^T M (Qv)}{(Qv)^T (Qv)} = \frac{v^T C v}{v^T v} = \frac{v_1^2 \lambda_1 + ... + v_n^2 \lambda_n}{v_1^2 + ... + v_n^2}$$ This ratio is a weighted average of the

eigenvalues, and is minimized by using just the smallest eigenvalue. We can see we get this if v is the eigenvector associated with the smallest eigenvalue of M).

So we have to find eigenvalues of the matrix on the left (M). *(D-W)* is called the Laplacian matrix, and is known to be positive semidefinite. So to minimize the generalized Rayleigh quotient, we need to pick the eigenvector with the smallest non-zero eigenvalue. When the eigenvalue is zero, the denominator of the ratio becomes 0, and the ratio is undefined.

We can verify that

$$z_0 = D^{\frac{1}{2}} 1$$

Is an eigenvector with eigenvalue zero, where **1** is a vector of all 1s, because

$$D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}} z_0 = D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}} D^{\frac{1}{2}} 1 = D^{-\frac{1}{2}} (D - W) 1$$

and (D-W)1 = 0 because W1 just counts up the number of edges leaving each node, and so is the same thing as D.

So, picking z to be the second smallest eigenvector, we minimize:

$$\frac{z^T D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}} z}{z^T z}$$

This is a basic Rayleigh quotient. Which means that the corresponding *y* minimizes:

$$\frac{y^T (D - W) y}{y^T D y}$$

So the continuous version of the normalized cut criterion is minimized by the eigenvector associated with the second smallest eigenvalue of

$$D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}}$$

Additional points:

y gives continuous values. Somehow, this has to be converted into two discrete regions. One way is to choose a threshold to separate the regions. This could be done to optimize the normalized cut criterion.

Finding the eigenvectors of D-W can be very expensive, since these matrices are nxn, where n is the number of pixels. It is important to make sure that W is a sparse matrix,

with most affinities set to zero, so we can use algorithms for finding eigenvectors of sparse matrices, which are much faster.