Math 123
Lecture Notes

Jan 19th

Linear algebra review:

Let $A \in \mathbb{R}^{m \times n}$, with entries $a_{i,j}$, $0 \leq i \leq m, 0 \leq j \leq n$. Of course, if $m = n$, A is square.

Recall that if $x \in \mathbb{R}^n$, we say that we can act on x via $Ax \in \mathbb{R}^m$, where $y = Ax$ has entries $y_i = \Sigma_{j=1}^n a_{i,j} x_j$

Somewhat obvious remark: Vector inner products can be realized as matrix multiplication, where we realize the inner product for $x, y \in \mathbb{R}^d$ as $\Sigma_{i=1}^d x_i y_i$. Alternatively, we may consider it as matrix multiplication of xy^T, where we realize x, y as $1 \times d$ matrices.

On the other hand, if we still realize these as $1 \times d$ matrices, we can call the outer product something like $x^T y \in \mathbb{R}^{d \times d}$.

Remark: if we have a rank 1 matrix, then we have a theorem that states that we can always decompose it as the outer product of two vectors.

Recall we may define $\|x\| = \sqrt{< x, x >}$

Recall that we may call the angle of two vectors as $\arccos\left(< x, y > / \|x\|^2 \|y\|^2\right)$

Recall that, for $A \in \mathbb{R}^{m \times n}$, we call $A^T \in \mathbb{R}^{n \times m}$, where $a_{i,j} = a_{j,i}^T$.

If $A^T = A$, then we call A symmetric. Note that of course, if A is symmetric, then A must be square.

Remark: If A is symmetric, then we may decompose A as:

$$A = U^T \Lambda U$$

where

(1) U is orthogonal/orthonormal (that is, $U^T U = U U^T = I$)

(2) Λ is diagonal

In particular, the values of Λ are the eigenvalue of the corresponding eigenvector in U.

Why would we care?

1) U orthogonal $\implies \{u_i\}_{i=1}^n \subset \mathbb{R}^n$ are linearly independent, that is, they span $\mathbb{R}^n$.

2) Thus, when we want to compute Ax, we may rewrite $x = \Sigma_{i=1}^n c_i u_i$

3) Thus, when we take $y = Ax$ rewritten in the eigenbasis, this is very easy: we can compute this easily as $y = \Sigma_{j=1}^n c_j \lambda_j u_j$

Jan 24th

Principal Component Analysis:

Suppose we have data points $\{x_i\}_{i=1}^n \subset \mathbb{R}^d$. We wish to find the directions of maximum variance in the data, and reduce the analysis into components $< D$, in order to capture as much of the variance in a lower dimensional subspace. In other words, we want to find low-dimensional structure in the data, generally when D is large.

We do this by taking a spectral value decomposition of the empirical covariance matrix.

We organize the data into a $n \times D$ matrix, where a row is a data point, and each column is a measurement of a variable across data points.

Given an orthogonal matrix $U \in \mathbb{R}^{n \times n}$, for any $x \in \mathbb{R}^n$, we may express $x = \sum_{i=1}^n c_i u_i$ where u_i is the i-th row/column of U. That is, the u_i span $\mathbb{R}^n$. However, this is not great for a generic matrix M, since this grows as the cube of the dimension. (Interesting point: solving a generic linear system grows as n^3 where n is the dimension). However, for an orthogonal matrix, we abuse the fact that $u_i \cdot u_j = \delta_{ij}$. We notice then that $Ux = (c_1, ..., c_n)$. (note that we can of course, compute each coefficient one by one by taking a dot product against each of the orthogonal vectors in turn) In particular, this has complexity n^2.

In any case, in this context, PCA is an effort to learn the "best" orthogonal matrix for my data. When we say best, we mean that if we were to take cutoffs, each choice of component preserves the maximum variance in the data, under the projection onto the component. Formally:

The projection of the data onto u_1 should be variance maximizing over all possible choices of u_1. More generally, we wish this to be variance maximizing over n components, for all n dimensional subspaces.

First, recall what variance means: for $x_1, ..., x_n \in \mathbb{R}$, we define the variance as:

$$\sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2, \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

where we call μ the mean.

Then, we need only compute the projection onto a direction $u \in \mathbb{R}^{D \times 1}$ by taking $u^T x$ for a data point $x \in \mathbb{R}^{D \times 1}$, and take the variance of these pure numbers. So, we wish to find:

$$\arg \max_{u \in \mathbb{R}^n, \|u\|=1} \frac{1}{n}\sum_{i=1}^{n}[u^T x_i - \mu]^2, \mu = \frac{1}{n}\sum_{i=1}^{n} u^T x_i$$

This kinda sucks. However, we claim that WLOG, that $\mu = 0$, because we can recenter the data at the origin by taking $\overline{x} = x - \mu$, where this μ is the mean of the original data points.

Well, let's now do some tricks:

$$(u^T x_i)^2 = (u^T x_i)(u^T x_i)^T = u^T x_i x_i^T u = u^T (x_i x_i^T) u$$

where we use the fact that since $u^T x_i$ is 1×1, so $(u^T x_i)(u^T x_i)^T$

Therefore, using the linearity:

$$\frac{1}{n}\sum_{i=1}^{n}[u^T x_i]^2 = \frac{1}{n}\sum_{i=1}^{n} u^T(x_i x_i^T)u = \frac{1}{n}u^T\left[\sum_{i=1}^{n}(x_i x_i^T)\right]u$$

We call

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i x_i^T)$$

the (empirical) covariance matrix of the data, where we understand $x_i \in \mathbb{R}^{D\times 1}$, i.e. a column vector.

In this case, we can do this via Lagrange multipliers.

$$\arg\max_{u\in\mathbb{R}^n, \|u\|=1} u^T\Sigma u - \lambda\left[uu^T - 1\right]$$

Differentiating with respect to u, we end up changing this into the case where

$$\frac{\partial}{\partial u}\left[u^T\Sigma u - \lambda(u^T u - 1)\right]$$

, where we claim that this is

$$2\Sigma u - 2\lambda u = 0 \implies \Sigma u = \lambda u$$

that is, u is an eigenvector of Σ.

Jan 26th

We recall that, given data $\{x_i\}_{i=1}^n \subset \mathbb{R}^d$, we wish to find the directions of maximum variance in the data such that $u_1 \perp u_2 \perp ... \perp u_d$, such that u_1 represents the direction of maximum variance, $\{u_1, u_2\}$ to be a plane, etc.

Recall this is an optimization problem as such:

Define $F(u) = \frac{1}{n}\sum_{i=1}^{n}(u^T x_i)^2$. Where we noticed that we could rewrite this as:

$$F(u) = \frac{1}{n}u^T\Sigma u$$

via a transpose trick.

After taking the derivatives, we recall that this becomes a eigenvalue equation. In particular, we notice that because this is symmetric, by the spectral theorem, that we have a D dimensional orthonormal basis from the eigenvectors.

Now, let's prove that the greedy algorithm works.

Suppose it works up to m vectors. Then, let's rewrite the condition for the $m+1$ vector as a Lagrange multiplier:

$$\mathcal{L}(u) = u^T\Sigma u - \sum_{i=1}^{m}\alpha_i(u^T v_i) - \alpha_{m+1}(uu^T - 1)$$

Taking a derivative again, we see that:

$$\frac{\partial}{\partial u}\mathcal{L}(u) = 2\Sigma u - \sum_{i=1}^{m} \alpha_i v_i - 2\alpha_{m+1} u = 0$$

Here, we look at this with respect to each v_i in turn.

$$v_j^T \left[2\Sigma u - \sum_{i=1}^{m} \alpha_i v_i - 2\alpha_{m+1} u \right] = 0 \implies 2v_j^T \Sigma u - \alpha_j - 2\alpha m + 1 v_j^T u = 0$$

Uh. I think James doesn't know what he's doing here.

Jan 31st

Redoing last time's proof.

Let Σ be a covariance matrix. Let $\{v_i\}_{i=1}^{D}$ be a set of orthonormal eigenvectors such that the corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_D$.

Suppose we know that the k dimensional hyperplane of greatest variance is spanned by the span of $\{v_i\}_{i=1}^{k}$. We wish to show that if we impose that u is perpendicular to v_i for $0 \leq i \leq k$, $u^T u = 1$, then $u = v_{k+1}$.

Writing this as a Lagrange multiple question:

$$\mathcal{L}(u) = u^T \Sigma u - \sum_{i=1}^{m} \alpha_i (u^T v_i) - \alpha_{m+1}(uu^T - 1)$$

Differentiating again, and setting to 0 such that this is a critical point:

$$\frac{\partial}{\partial u}\mathcal{L}(u) = 2\Sigma u - \sum_{i=1}^{m} \alpha_i v_i - 2\alpha_{m+1} u = 0$$

We claim that we may show that $\alpha_i = 0$ for $i \leq i \leq m$. If we multiply both sides by v_j^T for $1 \leq j \leq m$:

Then, since Σ is normal, and u is perpendicular to every v_j, we have that:

$$\alpha_j v_j^T v_j = \alpha_j = 0$$

Thus, we have that:

$$2\Sigma u = 2\alpha_{m+1} u \implies \Sigma u = \alpha_{m+1} u$$

Thus, u is an eigenvector, and to maximize the variance, we take an eigenvector with largest eigenvalue remaining.

What is this computational complexity of PCA?

Note that we usually just count scalar multiplications and additions and count them as equal in this toy model.

Well, suppose we have $\{x_n\}_{i=1}^{n} \subset \mathbb{R}^D$.

(1) Centering data: compute $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$. This has complexity nD

(2) Building $\Sigma = \frac{1}{n}\sum_{i=1}^{n} x_i^T x_i$. You can do some tricks, but it's of form nD^2

(3) Eigenvectors/values of Σ: of form D^3

So, we can say that overall, this has complexity of form $D^2(n + D)$, or $D^2 \max\{n, D\}$

Note that the eigendecomposition of Σ is available here because we have a symmetric, and thus normal matrix.

However, we can also do this for a matrix $A \in \mathbb{R}^{n\times d}$, for even $n \neq d$. We will lose potentially orthogonality though between left and right eigenvectors.

The singular value decomposition of a matrix $A \in \mathbb{R}^{n\times d}$ is the factorization:

$$A = U\Lambda V^T$$

such that Λ is "diagonal", $U \in \mathbb{R}^{n\times n}$, orthogonal, and $V^T \in \mathbb{R}^{d\times d}$, orthogonal.

More precisely, Λ has only non-0 entries when, if λ_{ij} represents the i-th row, j-th column, then λ_{ij} is non-0 only if $i = j$. Hence, we denote these as λ_i.

We call the columns of U the left singular vectors of A, and the rows of V^T the right singular vectors of A, and λ_i the singular values.

Feb 2nd

Talking more about SVD, we notice that it is unique up to scaling by norm 1 scalars, and permutation of pairs of vectors.

So one way we can think of this is that we can decompose an arbitrary matrix into a sum of rank 1 matrices:

$$A = \sum_{i=1}^{\sum\{n,d\}} \lambda_i u_i v_i^T$$

with a scale factor λ_i that corresponds to its singular value.

We recall that if A is square and symmetric, then this is very nice, since the scale factors are exactly eigenvalues, and the vectors $u_i, u_i = v_i$ are exactly eigenvectors.

Recall that for any matrix, the row rank is the dimension of the rows, and the column rank is the dimension of the columns, and the overall rank is the minimum of the two.

Theorem:

The rank of $A \in \mathbb{R}^{n\times d}$ is equal to the number of non-zero singular values.

Remark, a rank 1 matrix takes at least $n + d$ pieces of data, and more generally speaking, a rank k matrix has $k(n + d)$. This is nice, because the naive approach, you may need to allocate nd pieces of data.

PCA is connected to SVD as follows:

Recall we define $\Sigma = \sum x_i x_i^T$. Define $X \in \mathbb{R}^{n\times d}$ via the rows of X being x_i as row vectors.

We notice that $X^T X = \Sigma$, so PCA wants a SVD of $X^T X$.

So, let $X = U\Lambda V^T$, its SVD. Then, expanding $X^T X$:

$$X^T X = (U\Lambda V^T)^T (U\Lambda V^T) = V\Lambda^T U^T U\Lambda V^T = V\Lambda^T \Lambda V^T$$

Here, we notice that $\tilde{\Lambda} = \Lambda^T \Lambda$ is diagonal.

So, we have that $X^T X = V\tilde{\Lambda} V^T$, which is actually its eigendecomposition.

In summary, the eigendecomposition of Σ can be written in terms of the SVD of X. So we have a correspondence between the eigenvectors of Σ and the right singular vectors of X and between the square of the singular values to the eigenvalues.

The upshot here is that sometimes, taking the SVD of the rows/columns of your data points is a similarly efficient way to get the eigendecomposition instead of hitting the covariance matrix directly, but specifically, it's numerically more stable.

Feb 7th

Unsupervised Learning: Analysis of data without human-annoatated training labels.

Clustering is one example of this. We wish to assign labels to data, without training.

More concretely, suppose we have $\{x_i\}_{i=1}^{\infty} \subset \mathbb{R}^d$. Clustering will then assign labels $\{y_j\}$ for $1 \leq j \leq K$, and under some algorithm to determine the clusters, we assign x_i to each y_j.

The most foundational clustering method is K-means, where we try to find centers from the data, and minimize the distance, somehow, over these partitions.

So now consider a functional on the space of partitions.

$$F(C_1, ..., C_k) = \sum_{j=1}^{k} D(C_j)$$

where $D(C_j)$ is a shorthand for the sum of distances of points x_j in C_i to a privileged point.

Proposition:

Let $\{x_i\}_{i=1}^n \subset \mathbb{R}^D$. Define $G(x) = \frac{1}{n} \sum_{i=1}^n \|x_i - x\|^2$.

We notice that the mean minimizes G.

So here, for K-means, we choose the privileged point as the mean of a set of numbers.

So if we try to have $\mu_j = \frac{1}{|C_j|} \sum_{x_j \in C_j} x_j$, one function F can be:

$$F(C_1, ..., C_K) = \sum_{j=1}^{K} \sum_{x \in C_j} \|x - \mu_j\|^2$$

and thus, this becomes a minimization problem over the choice of partitions C_i.

But this is not easy to optimize, and it's not clear if this converges.

Scheme:

Input initial centroids $\{\mu_j\}$, with a tolerance $\epsilon = 1$.

Run an iteration. If the tolerance is more than ϵ, use centroids, assign points to the nearest centroid, and the then compute new centroids from these partitions.

If our functional has a delta from the last iteration of less than our tolerance, then we stop.

Otherwise, we continue, up to a maximum iteration count.

In general, we call this scheme a maximization expectation scheme. This specific one is Lloyd's Algorithm. This guy has an overall complexity in one iteration of order $nD + nKD$

Remark: to achieve convergence, you can potentially need $2^{\sqrt{n}}$ iterations.

Feb 9th

Recalling k-means, we notice that this is a hard assignment rule. That is, for each data point x_i, it gets exactly one label/cluster y_i. And maybe this is good, maybe this isn't ideal, and we want to include fuzziness.

Hierarchical Clustering instead gives a family of possible clusterings that relate to each other in a natural way.

Let $C = \{C_i\}_{i=1}^k$ be a clustering of data points $X = \{x_i\}$ into k clusters. That is, the C_i form a partition of X.

Let $\mathcal{B} = \{B_j\}_{j=1}^l$ be another partition of X. We call $\mathcal{B}$ nested inside of $\mathcal{C}$ if for each B_j, there exists a C_i such that $B_j \subseteq C_i$.

In particular, the family of clusterings that hierarchical clustering produces are exactly nested families. So the question here is, do you start with singletons or the whole set?

Definition:

We call a strategy that starts with singletons an agglomerative strategy. We call a strategy that starts with the full set divisive.

A remark: nesting structure is nice because we can visualize these via dendrograms, where a dendrogram is a tree that shows the splitting.

Ok, so what kind of merge rules make sense? The approach we will take is as follows:

(1) Define a distance between sets.

(2) Iteratively merge between the smallest distance sets.

So, the question is, how do we define this distance Δ? Two choices are as follows:

$$\Delta_{SL}(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|$$

which we call single linkage and

$$\Delta_{CL}(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\|$$

which we call complete linkage and

$$\Delta_{GA}(C_i, C_j) = \frac{\sum_{x \in C_i, y \in C_j} \|x - y\|}{|C_i||C_j|}$$

which we call group average.

Note, a naive single linkage algorithm is $O(n^3)$, since each comparison needs around n choose 2 comparisons, so on order n^2, and we have n iterations.

Theorem [Sibson 1975]:

You can do single linkage in $O(n^2)$.

Feb 14th

In summary, we've tried clustering based off of geometric shapes (K-means) as well as distances between clusters (hierarchical clustering).

Yet another approach is to consider a notion of density.

This tends to work well when we have many outliers or a lot of noise, since we can extract out what points are noise and what are data.

Conceptually, the idea would be to cluster things that are close to their 10 nearest neighbors, or some heuristic like that.

So, a foundational method is density-based spatial clustering of applications with noise, acronym-ed "DBSCAN".

So, we want to first identify and quarantine out the ouliers, based on how many close neighbors it has. Then, we group inliers together in an iterative manner.

Concretely:

Let $\{x_i\}_{i=1}^n \subset \mathbb{R}^D$.

For a point $x \in \mathbb{R}^D$, we take the closed ball $B_r(x) = \{y \in \mathbb{R}^d | d(x, y) \leq r\}$, for some metric. Unless we specify otherwise, we will assume this is the Euclidean distance.

Let $\epsilon > 0$, and define a min points between $1, n$. We take ϵ to be the threshold on how close a point needs to be to be a neighbor. We take min points to be the minimum number of neighbors within ϵ to be considered an inlier.

We call x a core point if $|B_\epsilon(x) \cap \{x_1, ..., x_n\}| \geq$ min points.

Call x a border point if x is not a core point, and there exists a core point z such that $x \in B_\epsilon(z)$.

Call x a noise point/outlier if it is neither a core nor a border point.

We then say that if x is directly density reachable from y if y is a core point, and $x \in B_\epsilon(y)$.

Further, we call x density reachable from y if there exists a sequence $\{x = x_0, x_1, ..., x_j = y\}$ such that x_{i+1} is directly density reachable from x_{i+1}.

We say x, y are density connected if there exists a core point z such that both x, y are density reachable from z.

Finally, the clusters learned by DBSCAN are maximal sets of density connected paths.

An example of when DBSCAN fails, but K-means works is a dumbbell. Everything is connected, but there are clearly two centroids. An example of the reverse is two concentric circles.

So good things: DBSCAN doesn't really care about the shapes or geometries of the situation.

Bad things, we have two parameters to tune, and it can be computationally expensive.

Remark: the notion of density is implicit in DBSCAN in terms of core, border, and noise points. We can more explicitly compute this via kernel-density estimates.

That is, let $\mathcal{K} : \mathbb{R}^d \to \mathbb{R}$ such that:
(a) $\mathcal{K}(x) \geq 0$
(b) $\int_{\mathbb{R}^d} K(x)dx = 1$
(c) $\mathcal{K}(x) = \mathcal{K}(-x)$.
We call such a function a kernel.

The kernel density estimator of our data is:

$$\rho(x) = \frac{1}{nh^d} \sum_{i=1}^{n} \mathcal{K}\left(\frac{x - x_i}{h}\right)$$

for some scale parameter h.

It turns out, with sufficient restrictions on n, h, that ρ converges to a nice function.

Feb 21st

A different family of approaches to unsupervised clustering work with graphs.

(1): Data comes in
(2): Construct a graph from the data
(3): Try to find communities in the graph, and use communities as clusters.

Define a graph $\mathbb{G}$ as a collection of vertices V and a collection of edges that connect vertices E, that is, $\mathcal{G} = (V, E)$. We will associate our data points as exactly vertices. The edges should encode how related our points are.

In practice, if we have n vertices, we can represent the edges as a matrix $n \times n$.

More precisely, we will consider "weighted graphs", that is, the edges have some value associated with them. Then, we store these values in a matrix $W \in \mathbb{R}^{n \times n}$.

Here, we will have W_{ij} be the weight between the i-th vertex and the j-th vertex, and we associate large weights with strong connections and vice versa.

A common choice is to use k nearest neighbors in the graph.

Example: fix an integer $k_{nn} \geq 1$. Define:

$$W_{ij} = \begin{cases} 1 & \text{if } x_i \text{ is a k-nearest neighbor of } x_j \text{ or vice versa} \\ 0 & \text{else} \end{cases}$$

Call x_i a k_{nn} nearest neighbor of x_j if $\|x_i - x_j\|$ is among the k_{nn} smallest distances in the collection of distances from x_j

Remark: an advantage is that this algorithm produces sparse matrices W, that is, most of its entries are 0.

This is nice because it has storage benefits since we don't need to initialize the zeros, and it's apparently faster to get the eigenvectors. This is not clear exactly why it works rigorously, but it works empirically.

Another example is if $W_{ij} = \mathcal{K}(x_i, x_j)$ where $\mathcal{K}$ is a kernel function.

In graph theory, we tend to use the heat kernel or Gaussian.

Define $\mathcal{K}(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$ for a tuning parameter σ.

Regardless, we can create a graph now. But how do we use it to determine communities? This ends up being difficult.

Attempt:

Partition a graph such that the weight of broken edges is minimal.

Define $F(C_1, C_2) = \sum_{x \in C_1} \sum_{y \in C_2} W_{xy}$

Then, we define the optimal partition as $(C_1^*, C_2^*) = \arg\min F(C_1, C_2)$

This is painful, since this is hard to compute. This also prefers cutting out singletons potentially.

Refinement:

We want the same thing, but we also want that C_1, C_2 have a similar size.

Define:

$$F(C_1, C_2) = \frac{\sum_{x \in C_1} \sum_{y \in C_2} W_{xy}}{\min\{\sum_i \sum_{j \in C_2} W_{ij}, \sum_{i \in C_1} \sum_j W_{ij}\}}$$

Feb 28th

Recall: Let $G = (V, E)$ be a weighted graph, with edge weights stored in a matrix. To be precise, let $V = \{x_1, ..., x_n\}$ be our points, with $x_i \in \mathbb{R}$, and let E be stored in a $n \times n$ matrix W. Typically, $W^T = W$, and $W_{ij} \geq 0$.

Example, we could construct a matrix W where $W_{ij} = \exp(-\|x_i - x_j\|^2/\sigma^2)$. But whatever, the point is we want to learn what a "good" weight computation is. We try to motivate the learning by cutting the graph, such that:

i) we break as few edges as possible

ii) the resultant subgraphs are roughly the same size

To achieve this, we try to minimize, over partitions C_1, C_2, the normalized cuts functional:

$$\text{Ncut}(C_1, C_2) = \frac{\sum_{i \in C_1, j \in C_2} W_{ij}}{\min\{\sum_{i \in C_1, j \in V} W_{ij}, \sum_{i \in C_2, j \in V} W_{ij}}$$

If we define the volume of a subgraph:

$$\text{Vol}(C_l) = \sum_{i \in C_l, j \in V} W_{ij}$$

We may rewrite this as:

$$\text{Ncut}(C_1, C_2) = \frac{\sum_{i \in C_1, j \in C_2} W_{ij}}{\min\{\text{Vol}(C_1), \text{Vol}(C_2)}$$

We notice that $\text{Vol}(C_1) + \text{Vol}(C_2)$ is approximately the total of the edge weights, minus the shared edges. Then, the minimum of the two is smallest when they are roughly equal.

So the denominator favors a balanced partition, and the numerator favors an unbalanced partition. This is hard to analyze.

Trying to simplify the situation, instead, let's define:

$$\text{Ncut}(C_1, C_2) = \sum_{i \in C_1, j \in C_2} W_{ij} \cdot \left(\frac{1}{\text{Vol}(C_1)} + \frac{1}{\text{Vol}(C_2)} \right)$$

which, we notice has a similar effect.

Remark, we can also use:

$$\sum_{i \in C_1, j \in C_2} W_{ij} \cdot \left(\frac{1}{|C_1|} + \frac{1}{|C_2|} \right)$$

for a similar effect, that is, using the cardinality of the partitions with respect to the nodes.

In any case, we want to find a partition C_1^*, C_2^* to be the partitions that minimize NCut. This turns out to be NP-hard.

The trick is to try to relax the condition, so that it's a smooth function, and try to take derivatives or similar. In general, this may not be possible.

Protocol:

1) Recast into linear algebra.

2) Relax the hard assignment on nodes being in exactly C_1 or C_2.

3) Use matrix calculus to find an approximate solution by eigendecomposition.

First, since this is a partition into two pieces, we can rewrite $(C_1, C_2) = (C, \overline{C})$, that is, its complement.

Define a vector $f^c \in \mathbb{R}^n$ via:

$$[f^c]_i = \begin{cases} \sqrt{\frac{\text{Vol}(\overline{C})}{\text{Vol}(C)}} & \text{if } i \in C \\ -\sqrt{\frac{\text{Vol}(C)}{\text{Vol}(\overline{C})}} & \text{if } i \notin C \end{cases}$$

Further, define

$$d_i = \sum_{j=1}^{n} w_{ij}$$

be the degree of x_i, that is, the sum of all edges that come from x_i.

Define the degree matrix $D \in \mathbb{R}^{n \times n}$ via:

$$D_{ij} = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

Thus, D is a diagonal matrix. Homework 6 preview:
i) Prove that sum of the vector entries of Df^c is 0.
ii) Prove that $[f^c]^T Df^c = \text{Vol}(G) = \sum_{i,j \in V} W_{ij}$
Define the graph Laplacian $L \in \mathbb{R}^{n \times n}$ via $L = D - W$.
It turns out, that we may connect L to NCut via:

$$f^{c^T} L f^c = \text{Vol}(G) \cdot \text{NCut}(C, \overline{C}) \implies \text{NCut}(C, \overline{C}) = \frac{1}{\text{Vol}(G)} f^{c^T} L f^c$$

We notice that f^c depends on C, the partition of course, but L is only dependent on the graph itself.

Then, we can reduce our original problem of determining C to minimizing this quantity over f^c. We haven't yet made progress, since these functions are differentiable, but our inputs are still rigid. Specifically, we notice that f^c has only countably many values, and we're not really defined on an open set then.

Well, relax the condition that f^c is actually from a partition, and just minimize over $\mathbb{R}^n$.

Thus, we want to minimize:

$$f^T L f$$

with the constraints that Df sums to 0 and $fDf = \text{vol}(G)$.

Turns out, since the objective is quadratic, and we have a linear, quadratic constraints, these tend to be bad. However, this turns out to be an eigenvalue problem on L.

Then, we can read off cluster values, by just taking $C = \{x_i | f > 0\}$ and $\overline{C}$ to be the complement.

March 2nd

Recall, for. a graph $G = (V, E)$, with associated weight matrix $W \in \mathbb{R}^{n \times n}$, we wish to partition the vertices into C_1, C_2 such that the following functional is minimized:

$$\text{Ncut}(C_1, C_2) = \sum_{i \in C_1, j \in C_2} w_{ij} \left(\frac{1}{\text{vol}(C_1)} + \frac{1}{\text{vol}(C_2)} \right)$$

This is NP-hard, but we relax the constraints and round instead, which is much easier.

Recall, let $d_i = \sum_{j=1}^n w_{ij}$ be the degree of node i. Let D be the degree matrix, with 0s and d_i. And finally, define $L = D - W$. Define:

$$[f^c]_i = \begin{cases} \sqrt{\frac{\text{Vol}(\overline{C})}{\text{Vol}(C)}} & \text{if } i \in C \\ -\sqrt{\frac{\text{Vol}(C)}{\text{Vol}(\overline{C})}} & \text{if } i \notin C \end{cases}$$

Then, we have the following properties:

(i) The sum of the vector entries of Df^c is 0.

(ii) $[f^c]^T Df^c = \text{Vol}(G) = \sum_{i,j \in V} W_{ij}$

(iii) ${f^c}^T L f^c = \text{Vol}(G) \cdot \text{NCut}(C, \overline{C})$

Thus, since $\text{Vol}(G)$ is a constant, we need only minimize ${f^c}^T L f^c$ over f^c.

From linear algebra, we have the result that, for a matrix M, if we wish to find either the argmax or argmin of $v^T Mv/\|v\|^2$ with respect to a vector v, then the argmax comes from the eigenvector with largest eigenvector, and argmin comes from the smallest.

Thus, this is a tractable problem. Turns out, we may take the solution that corresponds to an eigenvector of L.

Computational trick: Let $g = D^{1/2}F \implies F = D^{-1/2}g$ where $D_i = \sqrt{d_i}$, that is, the square root of the degree matrix.

Then, we look at:

$$g^T D^{-1/2} L D^{1/2} g$$

, specifically the argmin of g. The upshot here, is that our constraints are:

(i) $< g, D^{1/2}1 >= 0$

(ii) $\|g\|^2 = \text{Vol}(G)$

and here, we can define $L_{sym} = I - D^{-1/2} W D^{-1/2}$

So, we've reduced our problem to minimizing the quantity $g^T L_{sym} g$ with the above constraints.

But we know this. If we know that $D^{1/2}$ is not a eigenvector of L_{sym}, then g must be the smallest eigenvector of L_{sym}. Otherwise, it's the 2nd one.

So:

$$L_{sym}(D^{1/2}1) = D^{-1/2}(D - W)D^{-1/2}D^{1/2}1 = D^{-1/2}(D - W)1 = 0$$

Since we notice that $D1 - W1 = 0$, by the definition of the degree and weight matrices.

So, we have that $D^{1/2}1$ is an eigenvector of L_{sym} with eigenvalue 0, since L and thus, L_{sym} are positive semi-definite, we must have that the g that we want is the eigenvector with 2nd smallest eigenvalue. Call this value λ_2, with vector ϕ_2

Turns out, this has deep connections into the structure of the graph. $\lambda_2 > 0 \iff G$ is connected. Further, the magnitude of λ_2 is connected with the difficulty of partitioning the graph. And finally, back to our original problem, once we have ϕ_2, we choose $C = \{x_i : \phi_2(i) > 0\}, \overline{C} = \{x_i : \phi_2(i) \leq 0\}$.