1 Local Linear Embeddings (LLE)

For LLE, we had the following: For each data point x_i, we want to represent it based on its neighbors: $x_i \approx \sum_{j \in N(i)} w_j x_j$ (a linear combination of its neighbors). We'll optimize (minimize) the following objective: $\phi(w) = \sum_{i=1}^{N} \|x_i - \sum_{j \in N(i)} w_j x_j\|^2$. This is a reconstruction error enforcing a linearity assumption on the manifold around the near neighbors. We also require that $\sum_j w_j = 1$, with $w_j = 0$ if $j \notin N(i)$.

What is the point of all these algorithms? The point is to come up with a representation (embedding) of the data. This is only giving you part of the story: How are you related to your neighbors in a linear fashion. Once this relationship is learned, now what you can do is find in a lower dimension the mapping of the data points x_i and that respects exactly the same criteria.

Now, once you have learned the relationship, find the configuration of these data points in low dimensions which respects the same relationships: Now we have a low dim y, and we have already learned w. Now we optimize $\psi(y_i) = \|y_i - \sum_{j \in N(i)} w_{ij} y_j\|^2$. Then, this turns out to be minimizing the following objective:

$$\min \operatorname{Tr}(Y(1-w)(1-w)^T Y^T)$$

subject to $YY^T = I$. This now is an eigenvalue decomposition problem in the variable Y. We'll call $M = (1-w)(1-w)^T$.

However, it is not so good in practice.

2 Laplacian Eigenmaps

Our goal remains the same: Find a low-dimensional representation of our high-dimensional data, which preserves local structure in the data. LLE we had the data points be related to their near neighbors in a linear fashion. The question is how do you define local structure? The answer the authors of Laplacian Eigenmaps give is a little different. They will compute some sort of similarity between data points, and once you have a notion of similarity, they will optimize for position of the low-dimensional datapoints, which will keep the same similarity.

We will look at local distances as a measure of similarity. We'll then preserve this similarity in the embedding of the data points. Now the question is how the heck do you do that? If two things are more similar, you'll get a high value, if they're different you'll get a low value. Since we only know the position of data in space, what we can do is if the data points have a short distance, you want the value of similarity to be high and if they're far away, it should be small. It should be anti-proportional to distance.

What ends up getting used is a Gaussian type function.

Definition 2.1. Similarity.
$w_{ij} = \operatorname{sim}(x_i, x_j) = \exp(-\|x_i - x_j\|^2/\sigma^2)$ where σ is a scaling parameter.

But we're on a manifold, why are you talking about Euclidean distances and not geodesics? Well, there will be some sort of scaling parameter σ. If σ is at the same level as a local neighbor, then locally it'll be linear anyways, so you can compute the Euclidean distance as a good approximation. Outside the neighborhood, it'll be a geodesic distance. But since it's far away, the similarity will be small. But the Euclidean distance will *also* be small. Since the distance is dropping off quite sharply either way, it might be an ok approximation. σ is taken at the scale of "local neighbor". This is a mystery in itself. There might be a paper by Larry Wasserman which could tell you. Maybe different regions of space will have different notions of locality. This is based on some notion of curvature, so the question boils down to how you can estimate the curvature of a manifold. Little work has been done, but there is some work. That's where the state of the art is.

So now we have a notion of similarity. We want to come up with a configuration of our low-dimensional data points which will respect this similarity. So we have to come up with a cost function. For Laplacian Eigenmaps, it is the following cost:

Definition 2.2. Laplacian Eigenmaps Cost Function.

$$\psi(Y) := \sum_j w_{ij} \|Y_i - Y_j\|^2$$

We want to minimize this with respect to Y.

If w_{ij} is large, then Y_i and Y_j need to be close to each other. If they are dissimilar (e.g., $w_{ij} = 0$) then we don't care what happens. What will happen in that case? Then everything gets mapped to the same thing to minimize. So you have to add a condition: We require $Y^T Y = I$.

Now, this looks like a Graph Laplacian! This is $\text{Tr}(Y^T L Y)$, where L is the Graph Laplacian. In particular, it's $D - W$. D is the diagonal matrix, and W is the similarity matrix.

We can compare this to LLE. This is also the Laplacian! Since we have that the row sum of the W is 1, we end up getting a $I - W$ instead of $D - W$. So the solution type is the same for both of these methods.

There are a bunch of other algorithms.

3 Kernel Dimensionality Reduction

There is a unified framework for nonlinear dimensionality reduction via kernels. Essentially, all manifold embedding algorithms that exist in the literature are some sort of kernel PCA. Basically, we are just changing the kernel based on the particular instantiation.

What are we doing with PCA again? Finding a linear projection of data which is maximizing variance or minimizing distance to subspace by orthogonal projection. Kernel is saying there is a nonlinear transformation (maybe in high dimensions, maybe not) mapping ϕ which takes your data from the original space to another space nonlinearly. Now, we're going to do PCA in the transformed nonlinear space.

So one way of doing it is writing down SVD. Then $X = U\Sigma V^T$, and $XX^T = U\Sigma^2 U^T$, or the eigen-decomposition of the thing. $X^T X = V\Sigma^2 V^T$. Now the interesting thing is: What will this look like? The interesting thing is that the eigenvalues of the two matrices match, but the eigenvectors don't match. But there is a relationship between them. Let us note the following: In PCA, you're taking the higher-order spectra of the data point which you want to project to. So the data would look like $U^T x$, in PCA. If into k dimensions, you take $U_k^T x$. That is what we want to compute, but we only have the vectors V. But there is a relationship between them. From SVD, we have $U = XV\Sigma^{-1}$. Now if you have vectors V and Σ, you can compute this. Then you get, for a new data point, $\Sigma^{-1}V^T X^T x$. This is in terms of the inner product, which can now be generalized to kernels. So we take the kernel matrix $K = X^T X$. We can easily swap this out to our own kernel, instead of the linear inner product kernel. We just need some reasonably properties on K hold true (e.g., it needs to be PSD). That is, $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ for some nonlinear transformation ϕ, as encoded in the matrix K. The punchline is that all these nonlinear dimensionality reduction techniques (IsoMap, LLE, Laplacian Eigenmaps, and others) essentially apply implicitly some sort of a nonlinearity and doing PCA. So, the only thing that changes is K, because the nonlinearity changes. Here we list the various cases:

(a) Let's look at MDS (multi-dimensional scaling). We have $K = -\frac{1}{2}H^T DH$ where D is a Euclidean distance matrix and H turns it into a similarity. Essentially you're doing the same thing.

(b) As for IsoMap, we did exactly the same thing, but we'll change the distance matrix D to be the geodesic distance.

(c) In LLE, if we were to use $K = (I - W)(I - W)^T$. But now, we are looking at the lower order spectra, not the higher one. So we need to do some massaging to get the right kernel. Once W is known (it gets learned), then essentially we compute $A = (I - W)(I - W)^T$. We need to flip the spectrum of the matrix: We could do inverse and say $K = A^{-1}$. That's option one. Another way we could do it is define $K = \lambda_{\max} I - A$. Typically, this option gets used because inverses can be bad news in terms of stability. The point is that LLE is kernel PCA with this particular kernel.

(d) Laplacian Eigenmaps is even simpler. We have $A = D - W$. Then $K = \lambda_{\max} I - A$.

The point is to say that all these are special cases of kernel PCA from a different kernel. So (essentially) all these nonlinear dimension reduction methods are just a matter of finding the right kernel.

There's a reason I'm telling you this story. One point is that it unifies it, but that's not the main thing. The million dollar question is: Some of these methods give you one answer, and some give you another answer. So why can you not *learn* the right kernel, and see what happens. That gives rise to yet another nonlinear dimensionality reduction technique, which is called *Maximum Variance Unfolding* (MVU).

4 Maximum Variance Unfolding (MVU)

This used to be called Semidefinite Embedding (SDE). Why are we talking about this specific one? Multiple reasons. Our story began a certain way. We want to preserve local structure, this is what structure should be, here's an algorithm. Now we are not taking this recipe. What's the right way of defining local structure? MVU is by Lawrence Saul and one of his students Killian Weinberger. Around 2006, there was a lot of interest developed in terms of SDP optimization. So there were some advances made there, and machine learning people wanted to adopt those optimization techniques for machine learning problems. In terms of dimensionality reduction, you can formulate it as finding the positive semidefinite matrix itself. So there was a big rush at this time. In the early 90s, the rush was kernelizing every single method in machine learning. The mini-rush in 2006 was can we adopt the new technique of SDPs into machine learning, and this was one early incarnation where this happened in terms of nonlinear dimensionality reduction.

We're going to just formulate the problem, after it is an SDP, we just run SDP methods to get a kernel out. Then we can use kernel PCA. We need to formulate a problem to find the kernel. The question should be, what properties should hold for that particular kernel? What do you want for manifold learning? We'll list them out, and formulate it mathematically.

(a) Question 1: Say you want to preserve local geometry. What should hold true after applying the nonlinear transformation? How about nearest neighbors? That is, if $j \in N(i)$, we want the distances to match up: $\|\phi(x_i) - \phi(x_j)\| = \|x_i - x_j\|$. So this is one constraint. The problem is, it better be the case so that the RHS is a constant from data, and the LHS has a parameter ϕ. It had better be the case that distances can be written as dot products. This is the key constraint!

(b) Next time... what do you want for the non-neighbors to hold true? They should be far away. That's it! We'll maximize the variance subject to these kinds of constraints. This is basically doing PCA on the map.