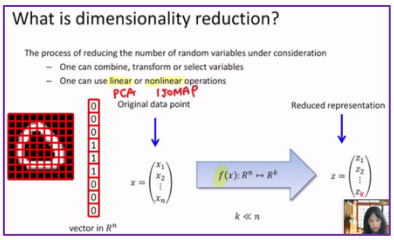
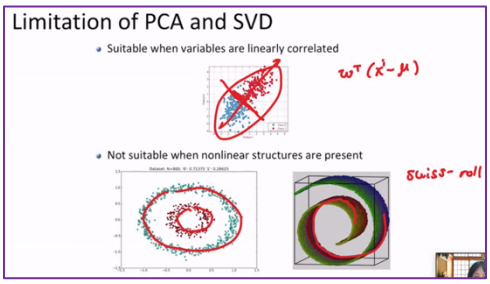
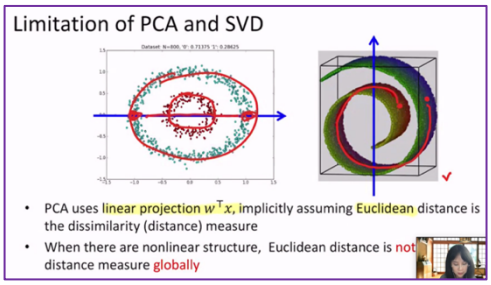
Hi, everyone. Welcome to today's lecture. We will talk about nonlinear dimensionality reduction. To recap, last time we talked about the problem of dimensionality reduction. The goal of this problem is to reduce the original high-dimensional data points into a much lower-dimensional representation. And last time we talked about a linear way to find this dimensionality reduction function, which is the **PCA** algorithm, and this time we're introducing another type of nonlinear way of constructing this map to reduce dimensionality, and this algorithm is called the **ISOMAP**.



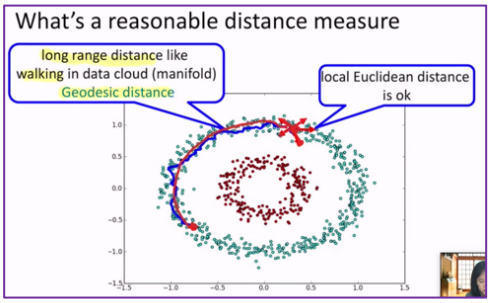
First of all, the question here is, why do we need to go beyond linear dimensionality reduction? PCA is derived based on linear projection. If you could remember that we're introducing these weight vectors combine the features, You have the $w^T(x^i - mu)$. We perform this linear projection on each of the data points. Essentially, this can only capture linear structure or linear dependence structure inside your data. And this works better if your data can indeed be captured through linear structure. For example, in this two-class dataset, the data are linearly separable. You can see if I project them in this direction, and I can separate them linearly using this boundary. However, this is typically insufficient when your data have nonlinear structure. When you have 2 rings, 2 clusters, or the data actually leaps on this structure it looks like a Swiss roll. This is called a Swiss roll dataset. In this case, the structure of the data, the geometry of data is not linear. If you try to project data points using linear projection, we were not going to preserve information and we'll destroy the structure in the data. For this dataset, we need to develop nonlinear dimensionality reduction methods to capture their structure.



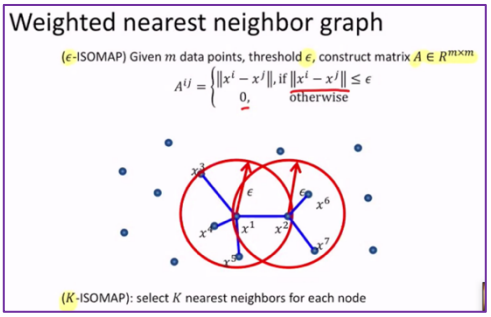
Here is a more precise summary. If you think about if you use linear dimensionality reduction on this dataset, what's going to happen? If I tried to project these two datasets using linear production along this direction, it's going to completely collapse the data into these clusters are green and somewhere in the middle and you couldn't identify that these data points actually leaps on two rings inside the 2-dimensional space, and similarly here for the Swiss roll dataset. If you think about more carefully, what is the problem associated with PCA? First, it uses these linear projections. Essentially, we captures the distance using Euclidean distance. That's the straight line between 2 data points. But when there is nonlinear structure, Euclidean distance cannot be sufficiently capturing this information. For example, if I want to measure the distance of one point here relative to another point here, if we look at their Euclidean distance, these 2 points are pretty close to each other. But this is not an accurate distance measure, because if you consider the geometry of the data, the distance of these two is better characterized by the travel distance to connect these two dots through the entire data clouds. How do we capture this generalized notion of distance considering data geometry is the first thing we'll consider here.



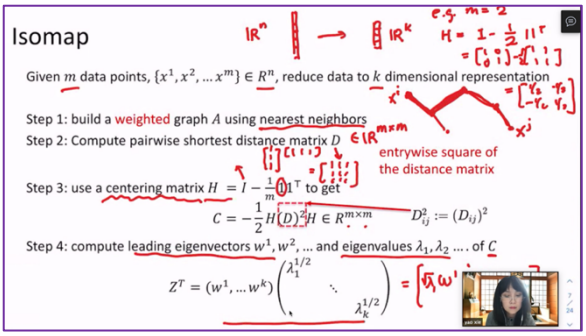
What is going to be a reasonable distance measure that can capture this? You can see if I consider these data clouds, and if you look at these data points, locally, Euclidean distance is probably fine because you can measure the distance of this point relative to its neighbor. This gives you accurate measure of distance. However, if you look at points that are farther away and I can't capture their accurate distance through Euclidean distance. But a more accurate measure would be how long it takes to travel through the structure of the data and travel through the day clouds from one point to the next. These long-range distances are more like walking distance or the distance it takes to connect the dust from one point to the other and so this is a notion called **geodesic distance** more generally. So essentially in nonlinear dimensionality reduction, we will utilize the geodesic distance to capture distance between the points and try to construct the projection based on that.



We'll start to introduce the algorithm. To start with, we will construct a weighted nearest neighbor graph. This version of the algorithm is going to be called **Epsilon ISOMAP**, two of the ISOMAP algorithms. And given m data points, we're going to send a threshold Epsilon, construct an adjacency matrix. We have discussed this before. Think about where you have seen this before. We have seen something like this in spectral clustering. But in spectral clustering, the nearest neighbor graph doesn't have weights. We're going to set A^{ij} equal to 1 when the two points are within Epsilon distance away from each other. This distance here, we're going to use the Euclidean distance or the L2 distance. But here ISOMAP we will consider a weighted nearest neighbor graph. A^{ij} equals to the Euclidean distance between the 2 points if they're within each other's neighborhood, Epsilon distance away. Otherwise, we're going to send it to 0. And there's another version of ISOMAP called the **K-**

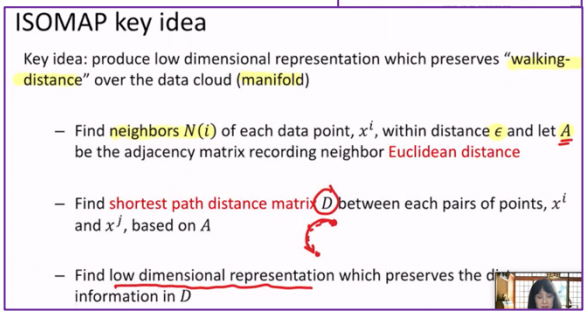


ISOMAP. The difference is instead of constructing the nearest neighbor graph using this approach, we're going to select a K-nearest neighbors. This will make sure each neighborhood will at least have K neighbor points. So, K-ISOMAP can typically work better with more sparse data set, meaning the points could be further away.

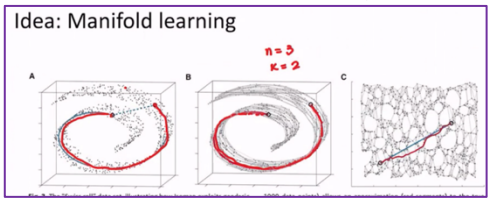


Now let's introduce the ISOMAP algorithm. Suppose there are initially m data points with n-D vectors, meaning they have n features. We're going to reduce them to a much lower dimensional representation with k dimensions. You can imagine this picture, pretty high dimensional, going to reduce them to a k-D. There are four steps involved. **Step 1** is we're going to build a weighted graph A using nearest neighbor. Here's a question to think about. Have you seen a nearest neighbor graph somewhere before? We talked about this in spectral clustering. But the difference is that the nearest neighbor graph there is not weighted, each edge is only 0 or 1, versus here we're going to use a weighted graph. **Step 2**, we'll compute pairwise shortest distance between any two data points. The intuition is the following, we have the nearest neighbor graph essentially, the data points are connected to their neighbors that are having a distance less than Epsilon, Epsilon far away. Now if I look at the distance between x^i and x^j data points, then their distance is going to be the total amount of distance traveled through this nearest neighbor graph. Now you can see why we need to keep the weights on this graph. While your capture the travel distance between two data points, we still need to keep each of the neighboring distance, we still need this distance information to calculate the total distance. The weight on these edges gives the local distance traveled between data points. We can do this for any pairs of data in our dataset that gives us the shortest distance D, a m-by-m square matrix. That ij is entry of the D gives the shortest distance traveled in this nearest neighbor graph. Step 3, we're going to introduce a special matrix **H**. Let me explain the notation a little bit. This I is the identity matrix, and this vector 1 is a vector of all '-1's, all the entries are 1, and the transpose of this all 1's. 11^T is a compact way of writing a matrix of all 1's. Let me write it here 11^T has all 1 entry. If you only have 2 data points, what is H is going to be? H is I – 1/2 11^T. This H (top right) performs the role of centering the data points, I'll explain later why this actually centers the data point. This H is used to construct a matrix C, $C = -1/2HD^2$. This notation is the entry wise square of the distance. This D^2, this matrix in the middle, is a matrix, and each entry of this matrix is the squared distance between any pairs of data. This C matrix basically captures the pairwise travelled distance of the data through the data clouds that's based on the shortest distance we have constructed. And the C matrix is dimension m by m. So now imagining this C matrix captures how data points are far away or close to each other in this data clouds. The last step, we'll compute eigen decomposition of C matrix and find the leading eigenvectors and eigenvalues. For example, if I'm going to reduce it to k-dimensional vectors, then my scores Z are going to be these vectors. I'm going to take the top k eigenvectors, these are the eigenvectors with the largest eigenvalues, then I'm going to multiply it by a diagonal matrix. On the diagonal, these are $\sqrt{\lambda}$. What happens here is if you perform this operation, essentially, these correspond to scale each of the vector by the square root of Lambda. If you can imagine, if this is my w^1, and this gives you $\sqrt{\lambda_1}w^1$, and this is the first, and then the second is $\sqrt{\lambda_2}w^2$ and so on.

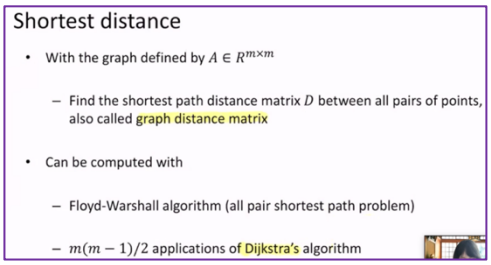
That's the description of the algorithm. Let's start explaining why the algorithm works like this. First, I like to say that actually this algorithm was developed in this paper by these three authors. And this paper, you can see clearly is considered to be a major breakthrough. It's published in the Science Magazine in 2000. And we're going to post this paper on Canvas as well. So hopefully you can read this yourself. It's a pretty short paper and it's a chance to get a feeling where this original works come from and then to look how they're presented in the original papers.



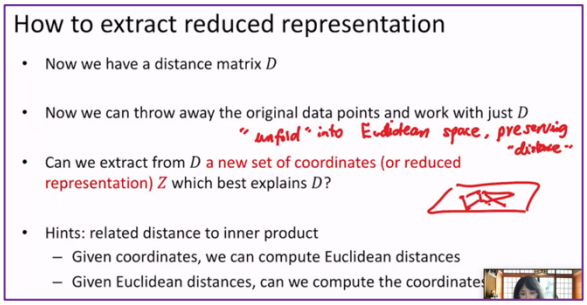
We're going to try to formulate it into a mathematical form. The key idea in ISOMAP is to produce low dimensional representation, which preserves the walking distance over the data clouds. And in the mathematical term, this data clouds actually have a name, and it's represented by a manifold geometry. You cannot measure long-term distance by Euclidean distance. But if you look at a very tiny little neighborhood around each data point, let's represent this as N(i), so this neighborhood is Epsilon distance away, just walk a little bit away from the data points. And then you can construct the geometry using Euclidean distance A. Based on that intuition, the nearest neighbor graph is going to be constructed using Euclidean distance. That is, you can understand we are performing local Euclidean approximation to this complex manifold. Once we have that, this local distance information is captured in the A. Then we'll try to find out the geodesic distance. Remember, we're saying the geodesic distance is, if you have a data culture look like this, you start from here if we're trying to get here, and then your geodesic distance is how much distance you have to travel on this data cloud along its geometry to reach to the other point. To get this estimate of the geodesic distance, we will **find** the shortest path distance matrix by travel through this nearest neighbor graph between any two points. That's the idea. Essentially, this shortest path matrix D captures geodesic distance. Then based on that, we're going to **find** low-dimensional representation which preserve the distance information in the D. And that will help us to unfold the manifold and understand it's intrinsic structure. That's the idea.



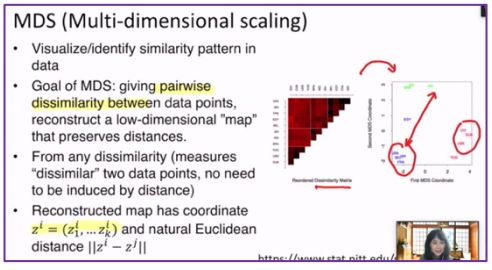
I should take this picture out of the paper. I think it illustrates the point. This is a Swiss roll dataset. If you look at these data points, each data point is a 3-D point, but they actually lie on a 2-D surface as if you have a piece of paper, you can roll it up, it forms this 2-D space row geometry. I want to measure the geodesic distance between this point and this point. Ideally, I want to estimate this distance here traveled through the data clouds. Using the nearest neighbor graph A and the shortest distance D, I can actually try to approximate the geodesic distance by the shortest path distance traveled through this data clouds. You can imagine it if I could unfold this manifold, just unfold this piece of paper from a three-dimensional structuring to the intrinsic dimension, which is 2. In this case actually n is 3, the intrinsic dimension is 2. If I could unfold it, then the data points are going to be unfold onto this 2-dimensional surface. My geodesic distance I want to get is going to be this line here in blue and then the red one is the shortest paths I find connecting the two points using data. I can get an estimate of the geodesic distance using the D matrix.



Now we have a graph defined by A. The question is, how do we find the shortest paths between any pairs of points? This is also called the **graph distance matrix**. There are many existing algorithms in computer science that solve this problem because it's a pretty important and well-studied problem. These algorithms include the Floyd-Warshall algorithm, which finds all pairs of shortest paths between points. And there's also Dijkstra's algorithm which solves the shortest distance between any two points in the graph, and you can apply this to all pairs that gives you the D matrix.



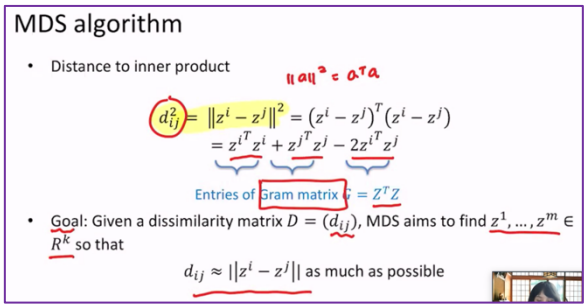
Now we are having this D matrix, essentially contains the travel or walking distance between any pairs of data points through your manifold geometry. In fact, I'm extrapolating, you can throw away the original data points and just work with just D. The question is the following. I'm imagining you have this manifold structure. You can capture the pairwise distance. This distance are connected through this manifold structure, this geodesic distance. Now I want to **unfold** and fold **into a Euclidean space** such that in this unfolding space, the distance between any pairs of points are going to be the same, it's going to be the same as D. So mathematically speaking, what does this mean? This means that you have n data points in a pretty high dimensional space. I can calculate their geodesic distance and then I'll try to flatten out my manifold for better understanding of what this data really looks like. I want to map these data points to a lower-dimensional space like a 2-D space, and the distance between them are going to be preserved. These distances are going to be the same as D, preserving distance between all pairs of data. That's basically the intuition. We want to map these data points in lower-dimensional Euclidean space while preserving the pairwise distance between them. So, to achieve this goal, we will solve the coordinate for each data points, new coordinate each data points in the Euclidean space such that these coordinates means where do I place the points where I'll preserve my distance. So how do I solve this problem? The hint is, let's try to relate distance to inner product between the coordinate vectors. Given the coordinates, we can compute the distance between the points. That's pretty easy. Now that it's an inwards problem, if I gave you Euclidean distance, can you find out the coordinate of these points?



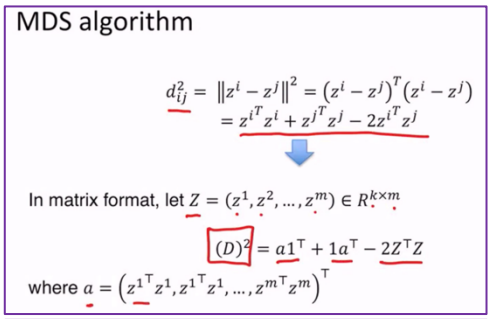
To solve this problem of placing the points given their pairwise distance, there is a pretty well studied algorithm called **MDS** (Multi-dimensional scaling). I'm going to talk about it because it's pretty neat and it's a fundamental building block in this manifold learning and non-linear dimensionality reduction. This is commonly used for visualizing, identifying similar patterns in data. The **goal** of MDS is exactly pairwise dissimilarities between data points. It could be distance, and in general, it can be any dissimilarity between data points, not need to be induced by distance. We want to reconstruct the coordinates, find a map, and that the distance between the points in this new coordinate, their Euclidean distance are going to be preserved. So here is one example, this is some dissimilarity score between pairs of countries like USA, China, France, and so on. Given this pairwise dissimilarity measure between countries, we use this algorithm. It maps countries in a 2-D plane as you can see. The mapping preserves their distance, meaning their similarity score, meaning that the points that are closer to each other will have a higher dissimilarity and the ones that are further away, these are less likely to be shown, as you can see. USA, Israel, and France are in this corner and then China, Cuba, USS are in this corner. It seems like make a lot of sense based on this mapping.

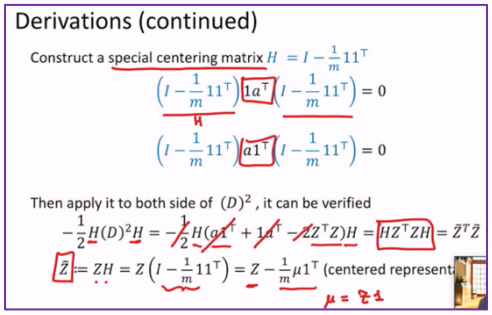
Let's see how we use **MDS algorithm** to achieve our goal. We are given these pairwise distance matrices and our goal is to relate it to the Euclidean distance between these reduced representations of the data points. We want d_{ij} = the Euclidean distance. Let's take the square on both sides, we have $d_{ij} = z_i - z_j$ cleaning distance squared. From here to here, we use a trick, the squared algorithm is A^TA, You end up having this, and let's break it into the products, You have this, this, and this is this, break them into the cross terms, and You have $z_i^Tz_i + z_j^Tz_j, - 2z_i^Tz_j$. What have you arrived from this derivation is that you will see, this seems like we need the inner product of these

z's, these vectors to satisfy this equation. All it matters in the end is the inner product between the z's, between the reduced reputations. In fact, these are all specific entries in the so-called **Gram matrix**. You'll see this a few times in lectures as well. Gram matrix is defined as pairwise inner product between some feature, here you can treat it, the features as this z. What is our goal here? We want to solve the following problem. Given the D matrix, we want to find these z's, which are dissertations such that this is true as much as possible. I'm saying this is approximate because in practice, of course, you're not going to have exact results because the data is always noisy and so we're trying to explain it as much as possible.

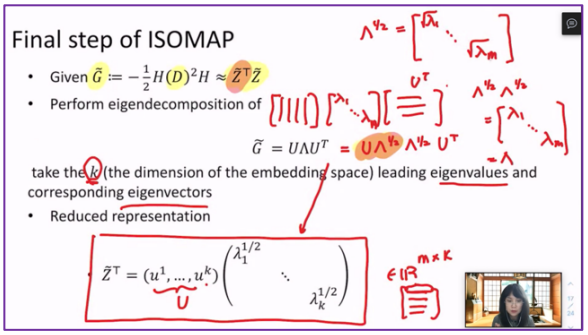


The first thing is, let's try to take this relationship we have find, then rearrange this in the matrix vector representation. We're going to define a matrix. This matrix Z contain all the low-dimensional representation of the data points. This Z matrix is a k-by-m matrix because each z is k-dimensional vector. You can verify that if I collect these d_{ij}^2 as entry of this matrix, then the right-hand side can be written as $a1^T + 1a^T - 2Z^TZ$, where the a is defined as this vector. So, now we'll introducing this special sensory matrix. Remember in this type of isometric, we have this matrix H. You can verify that this matrix have the following property. If you apply the H matrix on the left and right of this matrix, $1a^T$ or $a1^T$ is always equal to 0 and you can see this is going to be helpful because now if I apply this special matrix on the left and the right of D^2, then these terms are going to be knocked out and I will end up having this very simple last term. That's a pretty useful trick here, so I'm going to do this. I'm going to consider -1/2. This -1/2 is going to be at useful scaling factor in the end, H applied on left and right of D^2 matrix, and now I'm going to use the property, this is H. You have $Ha1^TH$, this term is going to be gone and same for this term. You end up having only the last term, so this -2 and -1/2 cancels, and I have HZ^TZH. If I can ratify a $Z^\sim$ (Tilda) to be ZH, so this term here can be written as $Z^{T}Z^\sim$. They have gone through a bunch of linear algebra, what does it mean here? If you look at the $Z^\sim$, what it is, is Z times. If you actually break it out, this original Z is something you want to find for these reduced representations, and this is Z – average. This mu is defined as Z1, so it essentially is computing the average of the reduced invitation along each of the dimensions. So essentially, you're subtracting the mean from the Z, alright, so this $Z^\sim$ are the Z removing the mean.

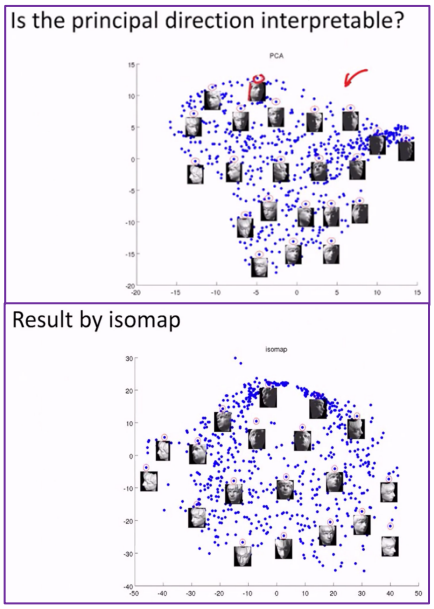




We're in the final step of ISOMAP. We are having this equation which tells us $G^\sim$, which depends on the data, the distance matrix constructed from data, and we want to find this, $Z^\sim$, which gave us the reduced representation of the data points. Once we find a Z, that gave us the desired result. So how do we solve this equation? $G^\sim = Z^{\sim T}Z^\sim$. We are going to use our Eigendecomposition. We can write $G^\sim$ as U which contains the eigenvectors of G and the Λ is a diagonal matrix that contains all the eigenvalues of G and U contains transpose, this is U^T here. Once we have this, we can write this as $U\Lambda^{1/2}\Lambda^{1/2}U^T$. Why is that the case? $\Lambda^{1/2}$ is a diagonal matrix where each entry is the square root of my original eigenvalue, so $\Lambda^{1/2}$ times itself because it might apply to diagonal matrix. Essentially, you're multiplying the corresponding diagonal entries that gives back. $\Lambda^{1/2}\Lambda^{1/2}$ equal to Λ. I have derived this. If I write it this way, as you can see, I can set my $Z^{\sim T}$ as $U\Lambda^{1/2}$. I'm going to set this to be equal to this. That gives one particular solution of this equation. Finally, this is our solution. We have found the $Z^\sim$, which is a particular low-dimensional representation of my data points, is given by this part. I'm going to extract in particular, not using the complete U, I'm only going to take the leading eigenvalues and corresponding eigenvectors because, remember, I want to map the thing approximately into a low-dimensional space and k is the dimension I want to embed it into. I'm going to only take the top k eigenvalue and eigenvectors. And this correspond to the approximation of U, only taking out the top k eigenvectors. And these are going to be the corresponding eigenvalues. That's it. After all of these you step back, you ask the question, what does it mean? Does this solution really make sense? The $Z^\sim$ is the embedding if you want it to be, but now you center this. I have to find the Z if you subtracted the center, and that gives you the tilde. We're pretty happy with $Z^\sim$ because translation of this data clouds doesn't really destroy the structure. The $Z^\sim$ is the final result we want. So, what is the dimension of the $Z^\sim$? The $Z^\sim$ is a matrix of m times k, m rows and k columns. And so essentially in this $Z^\sim$, it's going to look like you have m rows and k columns. Each row of my $Z^\sim$ gives me the coordinate of one data points. That's it. We have enough of our result. We have this ISOMAP algorithm, then, is this giving better result from PCA? We can also extract the leading eigenvectors that helps us to project the data into the leading eigenvectors of the covariance metrics, the principal components. If you only use 2 principal components, then we can represent each data point as a 2-D point, then we can plot it.

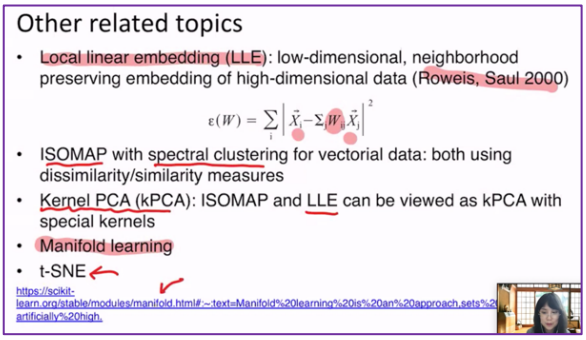


Let's compare PCA results versus what we have from ISOMAP by learning the geometry of the manifold and see why ISOMAP offers additional and useful information. And this is a dataset studied in the original paper for ISOMAP. This is a collection of images, as you can see this sculpture, their face are facing different directions. This(left) is the result using PCA. I can see for each point here that corresponds to one particular image. And this(right) is the result from ISOMAP. You can see results are quite different. The difference is, you can see it looks like in the ISOMAP, low-dimensional representation of these images, the images are organized in a more meaningful fashion. For example, you can see the faces are all facing to the left in this particular part and the faces are facing to the right in this particular part. They have the same orientation. The images have higher similarity, they are placed in one particular part of the space. So essentially this comes from some clustering pattern you want it to have that you can organize these images by their similarities. And there is a transition in orderly of these phases. From here to here, we can see the face seems to be gradually rotating from the left to the right. If you go down this direction, they're rotating to down and then they're rotating up. It's quite interesting that because when we solve ISOMAP, we have to find a local Euclidean distance, but then you have to automatically find the geodes of our distance traveled through the data clouds. The data points, you map them to the 2-D space, their relationship on the manifold are going to be preserved pretty well. That's why if you look at this manifold, it seems like make lot of sense, the faces are ordered in a meaningful way. But if you look at the PCA results, you don't quite see this pattern. There the face are organized in this meaningful interesting pattern.

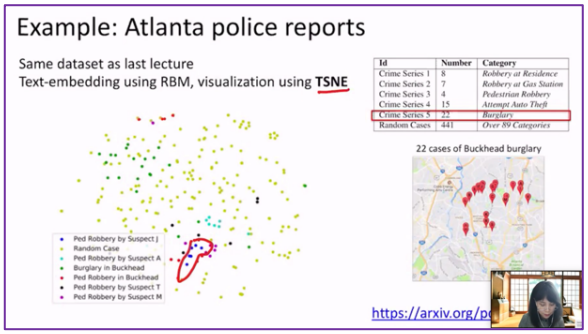


Here are more example also from the same science paper for ISOMAP. One is example for the hands. The results seems to be the hands that are doing one particular motion, is one direction and then doing another motion is on the other corner. And there seems to be natural transition from one end to the other end of the space. This mapping is interesting because you can see how roughly gives you the ordering based on the finger rotation and the wrist rotation. Similarly, this has been tried on the handwritten digits for number 2. This is a top arc rotation, bottom articulation. You can see the images seems to be organized in a pretty interesting and meaningful pattern.

The ISOMAP autism is one kind of algorithm and is a big class of algorithm that learns the non-linear dimensionality reduction. And also, manifold learning, basically you want to learn the geometry of the data instead of just using the patterns to describe them. There are other perform similar task, for example, local linear embedding, LLE, which is actually published in the same issue of Science after that article. You see the same problem arise at the same time on the same issue. And then you can see the LLE is formulated using a different formulation. The idea actually is quite intuitive. And also based on local geometry, you'll want to find a local representation of the data points using its neighbors. This can be solved to find these W's and that captures information about your data. ISOMAP actually can be viewed as a special kernel PCA. We talk about PCA, but PCA is linear. We can use kernel lives to transform PCA into a nonlinear principal component analysis. So, in fact, error E is also a type of kernel PCA. These correspond to some special can be shown. ISOMAP if you think about with spectral clustering based on this similarity measures and we talk about the spheres neighbor graph constructed from there. That's their connection. They're both based on the connectivity and geometry of the data instead of looking at clean distances.



Lastly, a pretty popular goal or tool to perform visualization and map the data to low-dimensional space is called the **t-SNE**. And here is an example on scikit-learn if want to create more about t-SNE. In fact, let me give you one example. We have used this in part of our research project to understanding geometry or connections between police reports data. And then we first perform a so-called **embedding**. Remember last time I described how do we represent the police report using the bag of words model. And each police report has a feature vector. Then the embedding is basically a machine learning algorithm that maps these data points. Each data point is police report into a space and then we want this imbedded. You have the property that it can put the similar crime incidences to be close to each other. You have this useful



clustering pattern. And to visualize embedding, we use the t-SNE algorithm mentioned before. I want to show you why this is interesting because in this plot, this is a result we get from the Atlanta police reports embedding. And the dots with the same color actually belong to the same crime series, meaning they're done by the same suspect. And You can see, quite interestingly, this machine learning algorithm is able to capture similarity between these cases. For example, all of these blue dots is a pedestrian robbery by suspect J, they have projected into a cluster in this embedding space using RBM restrict Boltzmann machine and t-SNE for visualization. And if you're interested in reading more, I posted the archive link down there.

In summary, today's lecture we have talked about a non-linear dimensionality reduction method called ISOMAP. And you can also view this as a manifold learning algorithm that helps us to understand, explore the non-linear geometry and structuring your data. It is quite useful, in particular for data exploration. And you're standing high-dimensional data. You can visualize high-dimensional data using this. Is different from the linear method, for example, PCA, in that I wrote the non-linear structure in your data and preserve the distance in a non-linear fashion. That's it for today's lecture. Thank you.