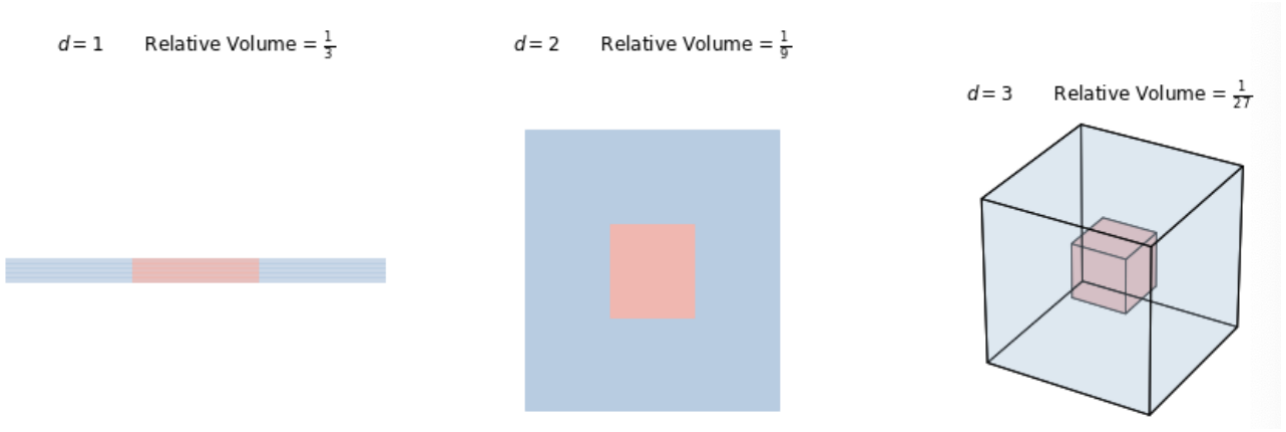
Dimensionality Reduction (PCA-MDS-ISOMAP-LE)

Trouble in High-Dimensional Land

In high-dimensional spaces, volume concentrates on the outside, exponentially more so, as dimension increases.Say we have two hypercubes, one of length l and another of $\frac{l}{3}$, what is the relative volume of the smaller cube to the larger cube? How does this proportion change as the dimension increases?



Relative Volume is given by:

$$\frac{(l/3)^d}{l^d} = (\frac{1}{3})^d$$

This implies that most of the volume in a hypercube lies around the edges (near the surface), and that very little volume lies in the center of the cube.

Difficulty 3: Distances

In PA lecture, this problem is described as the most samples that lie in the ϵ-boundary having similar intra-sample distances (probably because they are more dense). Therefore, **distances become less discriminative**. This problem also affects rejection sampling in high dimensions. Why?

Answer: the envelope distribution might be more dense in the outer regions and we may have to sample exponentially many times to hit the regions inside our target sampling distribution.

Difficulty 1: Visualization

Visualizing anything more than 3 dimensions is extremely difficult. As our data is usually 100, 1000 dimensional, we need to find (or *create*) the most important feature dimensions that can be easy to visualize.

Difficulty 2: Exponential # of bins

Similar features are at similar locations in the sample space. A classifier or a regressor must make local predictions. Assuming equally-sized cells, their number grows exponentially with

the dimensions. Hence, we require more model parameters, and moreover, and exponentially growing number of data points per cell

Therefore, we need dimensionality reduction techniques (linear or non-linear). In this lecture, we consider 4 algorithms:

1. PCA
2. MDS
3. ISOMAP
4. Laplacian Eigenmaps

Here is a meta-summary of the above 4 techniques:

PCA operates on explicitly given feature vectors x to maximize the spread of transformed data.
What if we only have access to differences between feature vectors, such as the Euclidean distance $||x_i - x_j||^2$?

MDS: This is exactly what MDS does. MDS tries to reconstruct a set of points (potentially in a lower dimension) from their differences (dissimilarity matrix)

ISOMAP: Essentially, MDS combined with **geodesic distance**, for reducing the dimensionality of data sampled from a smooth manifold. ISOMAP can handle Swissroll-type non-linear data where Euclidean distance is not appropriate. Therefore, it captures global geometry of the manifold

Weaknesses:

- very slow: need to compute pairwise shortest path between all sample pairs $O(n^3)$
- sensitive to "shortcuts".

ISOMAP

We often think that data is "inherently low-dimensional".
Images of a tea pot, taken from all images. Even though the images live in R^{256}, we believe they sit on a manifold corresponding to a circle:

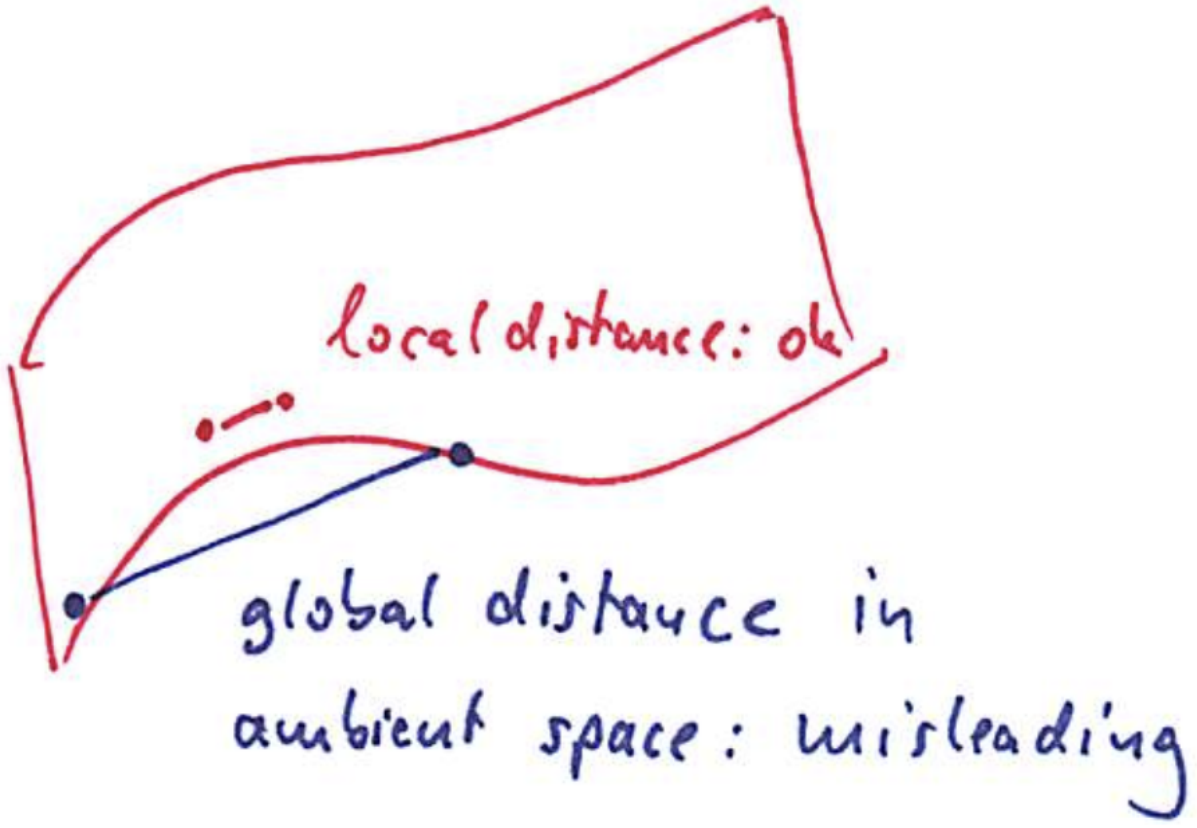
We would like to find a mapping that recovers this manifold.

Intuition about distances

In a small local region, Euclidean distances between points on a manifold approximately coincide with the intrinsic distances. **We want to keep the local distances unchanged**

However, this is no longer the case for large distances: instead we want to keep the

intrinsic(geodesic) distances rather the ones in ambient space.



If we want to straighten a manifold, we need to make sure that the Euclidean distance **after** embedding corresponds to the geodesic distance on the manifold.

How to calculate geodesic distances?

First, we need to **discover geodesic distances** in the manifold
To discover the geodesic distances in the manifold:

- Build a kNN graph on the data.
- Use the shortest path distance in the graph
- Idea is : it goes "along" the manifold

Remember k-NN graph is a graph in which two vertices p and q are connected by an edge, if the distance between p and q i**s among the k-th smallest distances** from p to other objects from P.

Algorithm

- Given some abstract data points $X_1, \ldots, X_n$ and a distance function $d(x_i, x_j)$.
- Build a k-nearest neighbor graph where the edges are weighted by the Euclidean distances to nearest samples. **These are the local distances.**
- In the kNN graph, compute the shortest path distances d_{sp} between all pairs of points and write them in the matrix D. **They correspond to the geodesic distances.** This

can be done via Floyd-Warshall or Dijkstra's algorithm
- Then apply metric MDS with D as input. Finds an embedding that preserves the geodesic distances.

As you can see, this last step is the same as MDS but we are providing a different distance matrix that preserves geodesic distances.

Tuning

K needs to be chosen carefully in k-NN graph. The idea is that k needs to be chosen so small that we follow the path in manifold in neighbor selection, but does not choose the points across the manifold (shortcut)

Theoretical Guarantees

- If the data points $X_1, \ldots, X_n$ are sampled uniformly from a “nice” manifold, then as $n \to \infty$ and $k \approx logn$, the shortest path distances in the kNN graph approximate the geodesic distances on the manifold.
- Attention, the dimension is an issue here. Typically, we cannot embed a manifold without distortion in a space of the intrinsic dimension, we need to choose the dimension larger

Laplacian Eigenmaps

The graph-based algorithms(ISOMAP, LLE, Laplacian Eigenmaps) have 3 basic steps:

1. Find K nearest neighbors or ϵ -ball neighbors
2. Estimate local properties of manifold by looking at neighborhoods found in Step 1
3. Find a global embedding that preserves the properties found in Step 2.

Euclidean distances between nearby points are transformed to **similarity scores**(to be used as weights) in one of the following ways:

- 0/1 weights: $w_ij = 1$ if there is an edge between x_i, x_j
- Gaussian weights: $w_{ij} = exp(-d_X(i,j)^2/t$ if there is an edge between x_i, x_j ($t > 0$ is a parameter to be selected by the user.) Heat kernel
  Each of such weighting methods leads to **similarity graph** with weights stored in a weight matrix $W = (w_{ij}) \in R^{n \times n}$

Let x'_i, x'_j be projected points in $R^{d'}$, and w_{ij} is an edge weight from above step. The objective function is then minimization of projected distances:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} (x'_i - x'_j)^2 w_{ij}$$

To make the objective function scaling invariant and also to get rid of the trivial solution 0 (if

we set $x_i = x_j$ we get the minimum too easily), we add the following constraint:

$$\tilde{x}D\tilde{x} = 1$$

where $\tilde{x} = (x'_1, \dots, x'_N)^T$ and $D = diag(\sum_{i=1}^{N} w_{ij}, \dots, \sum_{i=1}^{N} w_{Ni})$
Rewriting the objective function with this new notation, we get:

$$2(\tilde{x}^T(D - W)\tilde{x})$$

where one can replace L with $L = D - W$ (Laplacian). Using Lagrangian multiplier method, we can find the smallest eigenvalues along with corresponding eigenvectors.
However, we discard $\lambda = 0$ since they only indicate the number of independent components.
The lower-dimensional embedding is obtained by selecting an ordered subset of eigenvectors from that decomposition. This is similar to PCA embedding vector basis selection, but here we choose **smaller** eigenvectors and eigenvalues.

In short, the algorithm is the following:

1. Build an adjacency graph on the set of feature points $S = (x_1, \dots, x_n)$
2. Choose weights w_{ij} for the edges of the graph
3. Perform eigendecomposition of the graph Laplacian
4. Obtain low-dimensional embedding
   We choose weights based on affinities, meaning that closer point pairs have higher weights -> points that are closely together in high-dimensional space shall remain close in the lower-dimensional embedding. Two common choices for affinity calculation include binary affinity and gaussian heat kernel

Instead of using fixed edge weights, we can learn them via density forests:

1. Train a density forest
2. Calculate the sample affinity, aka relative frequency that both samples end up in the same leaf node of the trees.
   This brings the additional benefit that similarity measures better adapts the the data.

Laplacian attempts to approximate or preserve neighborhood information, while ISOMAP attempts to faithfully approximate all geodesic distances on the manifold.
Nearby points in the original space remain nearby in the reduced space.

In ISOMAP, a few outliers can create a shortcut on the manifold that significantly reduces the shortest paths.
In LE, all affinities jointly define closely the strongly connected components, hence isolated shortcuts have less impact.

PCA

Goal: given an **i.i.d** dataset $D = [x^{(1)}, \dots, x^{(N)}] \in R^D$. Our D is $N \times D$, find a mapping $U : R^D -> R^M$ such that $M < D$ that maximizes the variance (spread) of the data along

each dimension.
Objective function:

$$J = \sum_{i,j=1}^{N} (Ux_i - Ux_j)^T (Ux_i - Ux_j) + \lambda(U^T U - 1)$$

where x_i, x_j are data points. We want to maximize the J given the constraint that U is orthonormal
Covariance matrix S is obtained via:

$$S = \frac{1}{N} \sum_{i=1}^{N} (x_i - \tilde{x})(x_i - \tilde{x})^T$$

Then the variance of our data is given by :

$$u^T S u$$

We seek a unit-length direction **u** that maximizes the variance:

$$u^T S u + \lambda(1 - u^T u)$$

Taking the derivative with respect to u, we get the following eigen-value problem:

$$Su = \lambda u$$

Corresponding K eigenvectors u forms an orthogonal basis for our transformation matrix U. We can choose the number of components in U. Note that higher order eigenvectors explain (reconstruct) more parts of the data.
Interesting fact the dimensions of U are decorrelated.

$$\begin{aligned}
\mathbf{Cov}\left(\mathbf{z}\right) &= \mathbf{Cov}\left(\mathbf{U}^\top(\mathbf{x} - \boldsymbol{\mu})\right) \\
&= \mathbf{U}^\top \mathbf{Cov}\left(\mathbf{x}\right) \mathbf{U} \\
&= \mathbf{U}^\top \boldsymbol{\Sigma} \mathbf{U} \\
&= \mathbf{U}^\top \mathbf{Q} \boldsymbol{\Lambda} \mathbf{Q}^\top \mathbf{U} \\
&= \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} \boldsymbol{\Lambda} \begin{pmatrix} \mathbf{I} \\ \mathbf{0} \end{pmatrix} && \text{by orthogonality} \\
&= \text{top left } K \times K \text{ block of } \boldsymbol{\Lambda}
\end{aligned}$$

If covariance matrix is diagonal, then the features are decorrelated.
PCA is a linear method, which can be problematic in certain cases. That's why, one can use Kernel PCA to perform a non-linear dimensionality reduction

Noise in ISOMAP vs LE

- **Laplacian Eigenmaps**: Focus on preserving local neighborhood information. They construct a weight matrix based on the nearest neighbors of each data point and create a graph Laplacian that captures the local structure. Since the emphasis is on local relationships, Laplacian Eigenmaps **are less affected** by global distortions or noise in the data.
- **ISOMAP**: Focuses on preserving global geometric structure by approximating geodesic distances between all pairs of points using shortest paths on a graph. This global consideration can be sensitive to noise **because noisy data points can significantly alter the shortest paths**, thereby distorting the global structure of the manifold.