Lecture 19 - Mar 13

Dimensionality Reduction Beyond PCA
Multidimensional Scaling (MDS)
Graph-based Visualization
t-Stochastic Neighbor Embedding (t-SNE)

data visualization by optimization

References

- Chris Olah. Visualizing MNIST: An Exploration in Dimensionality Reduction (2014)

- *Elements of Statistical Learning*
  - 14.8 Multidimensional Scaling
  - 14.9 Nonlinear Dimension Reduction and Local MDS

- Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research* **9** 2579-2605 (2008)

Upcoming Deadlines

Homework 3 [Mar 15]

Midterm Exam [Take-home exam: Mar 14-16]
Material: Lectures 1-17
Topics: Regression and Classification

Multidimensional Scaling (MDS)

dataset: $x_1, \dots, x_n \in \mathbb{R}^d$ $(d>2)$

Goal: map each x_i to a point $z_i \in \mathbb{R}^2$ such that the distances between x_i's in $\mathbb{R}^d$ is similar to distances between corresponding z_i's in $\mathbb{R}^2$

(preserve geometry)

known

dist from x_i to x_j: $\|x_i - x_j\|_{2,d}$ — dimension

distance from z_i to z_j: $\|z_i - z_j\|_{2,2}$ — unknown

$$\text{Loss} = L(z_1, \dots, z_n) = \sum_{i \neq j} \left(\|x_i - x_j\|_{2,2} - \|z_i - z_j\|_{2,d} \right)^2$$

$2n$ unknowns

Minimize with numerical optimization

x_1, x_2 $\quad x_2, x_3$

x_1, x_3 $\quad x_3, x_4$

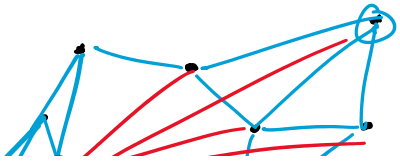
$\vdots$ $\quad \vdots$

x_3, x_5

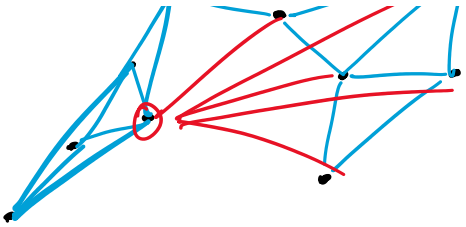
Focus more on local structure — Sammon's mapping helps

$$L(z_1, \dots, z_n) = \sum_{i \neq j} \frac{(\|x_i - x_j\| - \|z_i - z_j\|)^2}{\|x_i - x_j\|}$$

Graph-based visualization



for each point connect it to the 3 nearest



the 3 nearest neighbors

we shrink these $\Rightarrow$ ~~some~~ $\|z_i - z_j\|_{2,2}$

$$L(z_1, \dots, z_n) = \sum_{\substack{\text{all pairs} \\ x_i, x_j \text{ s.t.} \\ x_i, x_j \text{ are} \\ \text{connected}}} \left(\|x_i - x_j\|_{2,d} - \|z_i - z_j\|_{2,2} \right)^2 + \sum_{i \neq j} \frac{1}{\|z_i - z_j\|_{2,2}}$$

t-Stochastic Neighbors Embedding (t-SNE)

SNE (w t) is a Gaussian version

(1) It creates a ~~prob.~~ dist. of x_i's in $\mathbb{R}^d$

$$p_{i|j} = \frac{\exp\left(-\frac{\|x_i - x_j\|_{2,d}^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{-\|x_j - x_k\|^2}{2\sigma_k^2}\right)}$$

Assumes the L2 distances between points are normally distributed with mean of the point location & user-defined variance σ_i (hyperparameter)

(2) Constructs a dist. in $\mathbb{R}^2$

$$q_{i|j} = \frac{\exp\left(-\|z_i - z_j\|_{2,2}^2\right)}{\sum_{k \neq i} \exp\left(-\|z_j - z_k\|_{2,2}^2\right)}$$

SNE chooses z_i's to make these similar

Kullback-Leibler (KL) divergence measures how different two prob. distributions are

$$KL(P\|Q) = \sum_{i \neq j} p_{i|j} \ln\left(\frac{p_{i|j}}{q_{i|j}}\right)$$

minimize KL-divergence w.r.t. z_i's

Adjustment: Switch Gaussian to t-dist $\Rightarrow$ t-SNE

Multidimensional Scaling (MDS)

We start with observations $x_1, x_2, \ldots, x_N \in \mathbb{R}^p$, and let d_{ij} be the distance between observations i and j. Often we choose Euclidean distance $d_{ij} = ||x_i - x_j||$, but other distances may be used. Further, in some applications we may not even have available the data points x_i, but only have some *dissimilarity* measure d_{ij} (see Section 14.3.10). For example, in a wine tasting experiment, d_{ij} might be a measure of how different a subject judged wines i and j, and the subject provides such a measure for all pairs of wines i, j. MDS requires only the dissimilarities d_{ij}, in contrast to the SOM and principal curves and surfaces which need the data points x_i.

Multidimensional scaling seeks values $z_1, z_2, \ldots, z_N \in \mathbb{R}^k$ to minimize the so-called *stress function*[13]

$$S_M(z_1, z_2, \ldots, z_N) = \sum_{i \neq i'} (d_{ii'} - ||z_i - z_{i'}||)^2. \qquad (14.98)$$

This is known as *least squares* or *Kruskal–Shephard* scaling. The idea is to find a lower-dimensional representation of the data that preserves the pairwise distances as well as possible. Notice that the approximation is in terms of the distances rather than squared distances (which results in slightly messier algebra). A gradient descent algorithm is used to minimize S_M.

Optimization-Based Dimensionality Reduction

56

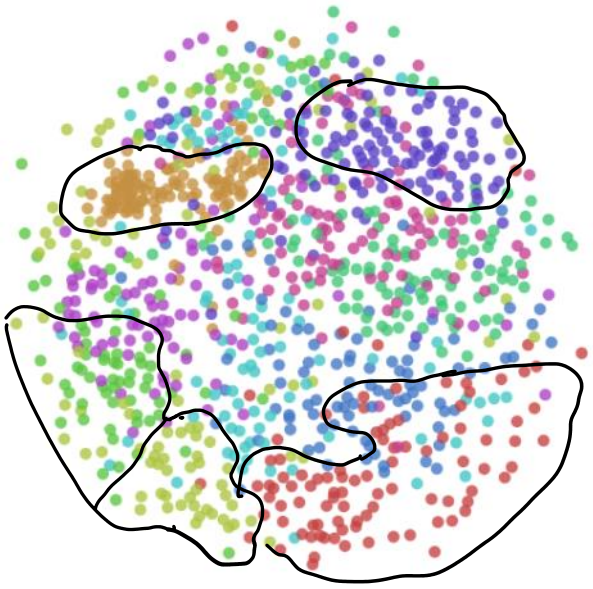
What would we consider a success? What would it mean to have the 'perfect' visualization of MNIST? What should our goal be?

One really nice property would be if the distances between points in our visualization were the same as the distances between points in the original space. If that was true, we'd be capturing the global geometry of the data.

Let's be a bit more precise. For any two MNIST data points, x_i and x_j, there are two notions of distance between them. One is the distance between them in the original space[1] and one is the distance between them in our visualization. We will use $d^*_{i,j}$ to denote the distance between x_i and x_j in the original space and $d_{i,j}$ to denote the distance between x_i and x_j in our visualization. Now we can define a *cost*:

$$C = \sum_{i \neq j} (d^*_{i,j} - d_{i,j})^2$$

This value describes how *bad* a visualization is. It basically says: "It's bad for distances to not be the same. In fact, it's quadratically bad." If it's high, it means that distances are dissimilar to the original space. If it's small, it means they are similar. If it is zero, we have a 'perfect' embedding.



Sammon mapping

$$L(z_1,\dots,z_n) = \sum_{i\neq j} \frac{(\|x_i-x_j\|_2 - \|z_i-z_j\|_2)^2}{\|x_i-x_j\|_2}$$

if x_i + x_j are close, denominator is small
⇒ loss is big

This technique is called multidimensional scaling (http://en.wikipedia.org/wiki/Multidimensional_scaling) (or MDS). If you like, there's a more physical description of what's going on. First, we randomly position each point on a plane. Next we connect each pair of points with a spring with the length of the original distance, $d^*_{i,j}$. Then we let the points move freely and allow physics to take its course!

We don't reach a cost of zero, of course. Generally, high-dimensional structures can't be embedded in two dimensions in a way that preserves distances perfectly. We're demanding the impossible! But, even though we don't get a perfect answer, we do improve a lot on the original random embedding, and come to a decent visualization. We can see the different classes begin to separate, especially the ones.

==========

A variation on least squares scaling is the so-called *Sammon mapping* which minimizes

$$S_{Sm}(z_1, z_2, \dots, z_N) = \sum_{i\neq i'} \frac{(d_{ii'} - \|z_i - z_{i'}\|)^2}{d_{ii'}}. \qquad (14.99)$$

Here more emphasis is put on preserving smaller pairwise distances.

Like the self-organizing map and principal surfaces, multidimensional scaling represents high-dimensional data in a low-dimensional coordinate system. Principal surfaces and SOMs go a step further, and approximate the original data by a low-dimensional manifold, parametrized in the low dimensional coordinate system. In a principal surface and SOM, points close together in the original feature space should map close together on the manifold, but points far apart in feature space might also map close together. This is less likely in multidimensional scaling since it explicitly tries to preserve all pairwise distances.

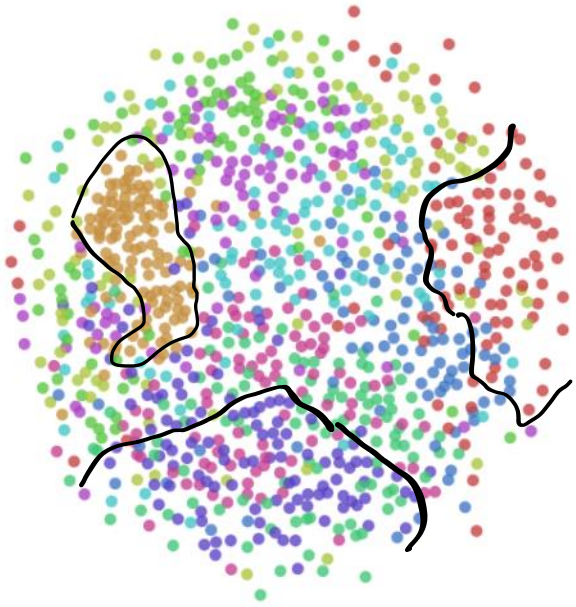
==========

Sammon's Mapping

Still, it seems like we should be able to do much better. Perhaps we should consider different cost functions? There's a huge space of possibilities. To start, there's a lot of variations on MDS. A common theme is cost functions emphasizing *local* structure as more important to maintain than global structure. A very simple example of this is Sammon's Mapping (http://en.wikipedia.org/wiki/Sammon_mapping), defined by the cost function:

$$C = \sum_{i \neq j} \frac{(d^*_{i,j} - d_{i,j})^2}{d^*_{i,j}}$$

In Sammon's mapping, we try harder to preserve the distances between nearby points than between those which are far apart. If two points are twice as close in the original space as two others, it is twice as important to maintain the distance between them.



For MNIST, the result isn't that different. The reason has to do with a rather unintuitive property regarding distances in high-dimensional data like MNIST. Let's consider the distances between some MNIST digits. For example, the distance between the similar ones, and , is

$$d(\,\text{}\,,\,\text{}\,) = 4.53$$

On the other hand, the difference between the very different data points,  and  , is

$$d(\,\text{}\,,\,\text{}\,) = 12.0$$

less than three times $d($, $)$!

Because there's so many ways similar points can be slightly different, the average distance between similar points is quite high. Conversely, as you get further away from a point, the amount of volume within that distance increases to an extremely high power, and so you are likely to run into different kinds of points. The result is that, in pixel space, the difference in distances between 'similar' and 'different' points can be much less than we'd like, even in good cases.

Several methods have been recently proposed for nonlinear dimension reduction, similar in spirit to principal surfaces. The idea is that the data lie close to an intrinsically low-dimensional nonlinear manifold embedded in a high-dimensional space. These methods can be thought of as "flattening" the manifold, and hence reducing the data to a set of low-dimensional coordinates that represent their relative positions in the manifold. They are useful for problems where signal-to-noise ratio is very high (e.g., physical systems), and are probably not as useful for observational data with lower signal-to-noise ratios.

The basic goal is illustrated in the left panel of Figure 14.44. The data lie near a parabola with substantial curvature. Classical MDS does not preserve the ordering of the points along the curve, because it judges points on opposite ends of the curve to be close together. The right panel shows the results of *local multi-dimensional scaling*, one of the three methods for non-linear multi-dimensional scaling that we discuss below. These methods use only the coordinates of the points in p dimensions, and have no other information about the manifold. Local MDS has done a good job of preserving the ordering of the points along the curve.

We now briefly describe three new approaches to nonlinear dimension reduction and manifold mapping.

Isometric feature mapping (ISOMAP) (Tenenbaum et al., 2000) constructs a graph to approximate the geodesic distance between points along the manifold. Specifically, for each data point we find its neighbors—points within some small Euclidean distance of that point. We construct a graph with an edge between any two neighboring points. The geodesic distance between any two points is then approximated by the shortest path between points on the graph. Finally, classical scaling is applied to the graph distances, to produce a low-dimensional mapping.

Local linear embedding (Roweis and Saul, 2000) takes a very different approach, trying to preserve the local affine structure of the high-dimensional data. Each data point is approximated by a linear combination of neighboring points. Then a lower dimensional representation is constructed that best preserves these local approximations.

Local MDS (Chen and Buja, 2008) takes the simplest and arguably the most direct approach. We define $\mathcal{N}$ to be the symmetric set of nearby pairs of points; specifically a pair (i, i') is in $\mathcal{N}$ if point i is among the K-nearest neighbors of i', or vice-versa. Then we construct the stress function

$$\begin{aligned} S_L(z_1, z_2, \ldots, z_N) &= \sum_{(i,i')\in\mathcal{N}} (d_{ii'} - ||z_i - z_{i'}||)^2 \\ &\quad + \sum_{(i,i')\notin\mathcal{N}} w \cdot (D - ||z_i - z_{i'}||)^2. \quad (14.105) \end{aligned}$$

Here D is some large constant and w is a weight. The idea is that points that are not neighbors are considered to be very far apart; such pairs are given a small weight w so that they don't dominate the overall stress function. To simplify the expression, we take $w \sim 1/D$, and let $D \to \infty$. Expanding (14.105), this gives

$$S_L(z_1, z_2, \ldots, z_N) = \sum_{(i,i')\in\mathcal{N}} (d_{ii'} - ||z_i - z_{i'}||)^2 - \tau \sum_{(i,i')\notin\mathcal{N}} ||z_i - z_{i'}||, \quad (14.106)$$

where $\tau = 2wD$. The first term in (14.106) tries to preserve local structure in the data, while the second term encourages the representations $z_i, z_{i'}$ for pairs (i, i') that are non-neighbors to be farther apart. Local MDS minimizes the stress function (14.106) over z_i, for fixed values of the number of neighbors K and the tuning parameter τ.

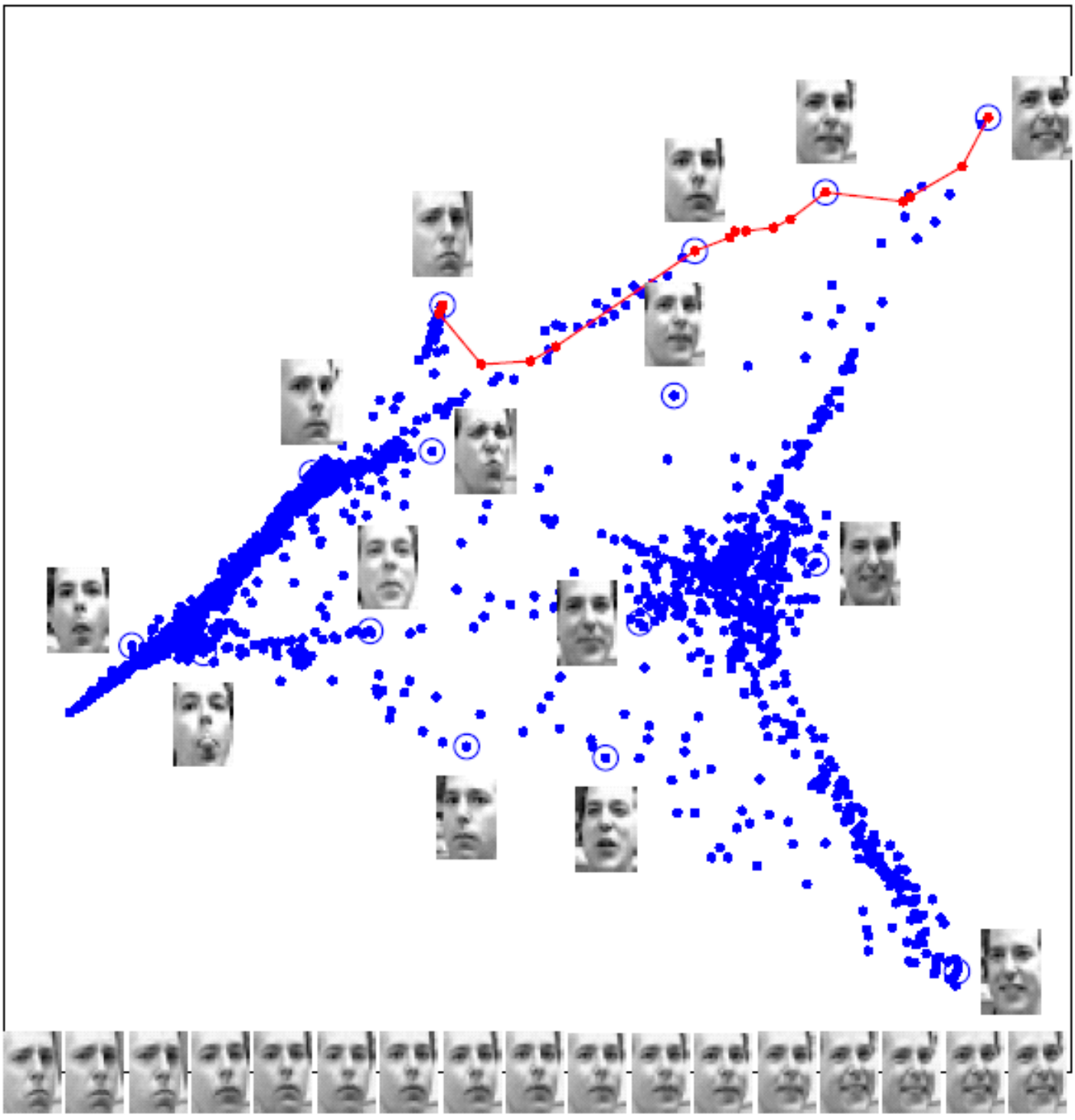


FIGURE 14.45. *Images of faces mapped into the embedding space described by the first two coordinates of LLE. Next to the circled points, representative faces are shown in different parts of the space. The images at the bottom of the plot correspond to points along the top right path (linked by solid line), and illustrate one particular mode of variability in pose and expression.*

Graph-based Viz

Graph Based Visualization

Perhaps, if local behavior is what we want our embedding to preserve, we should optimize for that more explicitly.

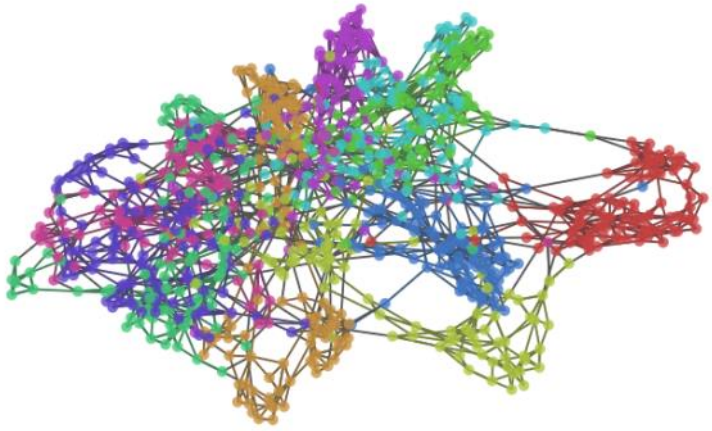
Consider a nearest neighbor graph (http://en.wikipedia.org/wiki/Nearest_neighbor_graph) of MNIST. For example, consider a graph (V, E) where the nodes are MNIST data points, and each point is connected to the three points that are closest to it in the original space.[3] This graph is a simple way to encode local structure and forget about everything else.

Force-directed graph drawing algorithms are a class of algorithms for drawing graphs in an aesthetically-pleasing way. Their purpose is to position the nodes of a graph in two-dimensional or three-dimensional space so that all the edges are of more or less equal length and there are as few crossing edges as possible, by assigning forces among the set of edges and the set of nodes, based on their relative positions, and then using these forces either to simulate the motion of the edges and nodes or to minimize their energy.

Given such a graph, we can use standard graph layout algorithms to visualize MNIST. Here, we will use force-directed graph drawing (http://en.wikipedia.org/wiki/Force-directed_graph_drawing): we pretend that all points are repelling charged particles, and that the edges are springs. This gives us a cost function:

$$C = \sum_{i \neq j} \frac{1}{d_{i,j}} + \frac{1}{2} \sum_{(i,j) \in E} (d_{i,j} - d^*_{i,j})^2$$

Which we minimize.



The graph discovers a lot of structure in MNIST. In particular, it seems to find the different MNIST classes. While they overlap, during the graph layout optimization we can see the clusters sliding over each other. They are unable to avoid overlapping when embedded on the plane due to connections between classes, but the cost function is at least *trying* to separate them.

One nice property of the graph visualization is that it explicitly shows us which points are connected to which other points. In earlier visualizations, if we see a point in a strange place, we are uncertain as to whether it's just stuck there, or if it should actually be there. The graph structure avoids this. For example, if you look at the red cluster of zeros, you will see a single blue point, the six , among them. You can see from its neighbors that it is supposed to be there, and from looking at it you can see that it is, in fact, a very poorly written six that looks more like a zero.

t-SNE

t-Distributed Stochastic Neighbor Embedding

The final technique I wish to introduce is the t-Distributed Stochastic Neighbor Embedding (http://jmlr.csail.mit.edu/papers/volume9/vandermaaten08a/vandermaaten08a.pdf) (t-SNE). This technique is extremely popular in the deep learning community. Unfortunately, t-SNE's cost function involves some non-trivial mathematical machinery and requires some significant effort to understand.

But, roughly, what t-SNE tries to optimize for is preserving the *topology* of the data. For every point, it constructs a notion of which other points are its 'neighbors,' trying to make all points have the same number of neighbors. Then it tries to embed them so that those points all have the same number of neighbors.

In some ways, t-SNE is a lot like the graph based visualization. But instead of just having points be neighbors (if there's an edge) or not neighbors (if there isn't an edge), t-SNE has a continuous spectrum of having points be neighbors to different extents.

t-SNE is often very successful at revealing clusters and subclusters in data.

https://medium.com/engineer-quant/t-sne-the-bits-that-no-one-learns-b5ce959ea1c2

...

This is indeed why there is a need for dimensionality reduction, that not only reduces the dimension of the dataset to make analysis easier, but also preserves the nature and structure of the dataset. A class of dimensionality reduction algorithms that exploit the geometry of the data is known as manifold learning. For an overview on manifold learning, do look into this article: Manifold Learning

In this article, I will be discussing in depth the t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm which is under manifold learning and is incredibly useful to visualize high dimensional datasets. It is my humble opinion that before using analysis tools, it is important to understand the theoretical background behind it and the limitations that results from the tool. Therefore, I will delve into the workings of the algorithm and look a bit into how it is implemented.

t-SNE

T-distributed Stochastic Neighbor Embedding, or t-SNE as it is normally called, is a manifold learning algorithm that in essence constructs a probability distribution over the dataset, and then another probability distribution in a lower dimensional data space, making both the distribution as 'close' as possible (we will make this notion more rigorous later).

However, before looking into t-SNE, we will investigate SNE, which uses Gaussian distribution instead of Student-t distriution. The first step is constructing the probability distribution of the dataset in high dimensions. The probabilities in the dataset are constructed using the Gaussian distribution which amounts to

$$p_{i|j} = \frac{e^{\frac{-||x_i - x_j||^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{\frac{-||x_i - x_k||^2}{2\sigma_i^2}}}$$

The intuition behind this construction is rather simple. The Euclidean distances between the pairwise points are considered to be normally distributed with mean of the point location and variance sigma, which the user defines.

Once the probability distribution in the high dimension is constructed, we construct a probability distribution in the lower dimensional space (usually 2 or 3 for visualization purposes) using the following formula for pairwise points y.

$$q_{i|j} = \frac{e^{-||y_i - y_j||^2}}{\sum_{k \neq i} e^{-||y_i - y_k||^2}}$$

Now the goal is to make sure the two probability distributions are as similar as possible. This is achieved by considering the Kullback-Leibler (KL) divergence.

KL divergence is a measure of how different two probability distributions are from one another. The KL divergence is defined as

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \ln \left(\frac{p_{ij}}{q_{ij}} \right)$$

In essence, the KL divergence is the expected value of the log difference of the probabilities of the data points. Now that we have a notion of how to measure the similar/difference between distributions, we can optimize this by well known gradient descent methods.

However, there is a crucial disadvantage to using SNE. It is called the crowding problem and it is a result of the curse of dimensionality. Consider that you have some data points that have ten intrinsic dimensions, but represented in a much higher dimensional space and you want to reduce it to two dimensions. It is possible that if the number of data points is little, say 11, then there is no reliable way to map the data points in two dimensions.

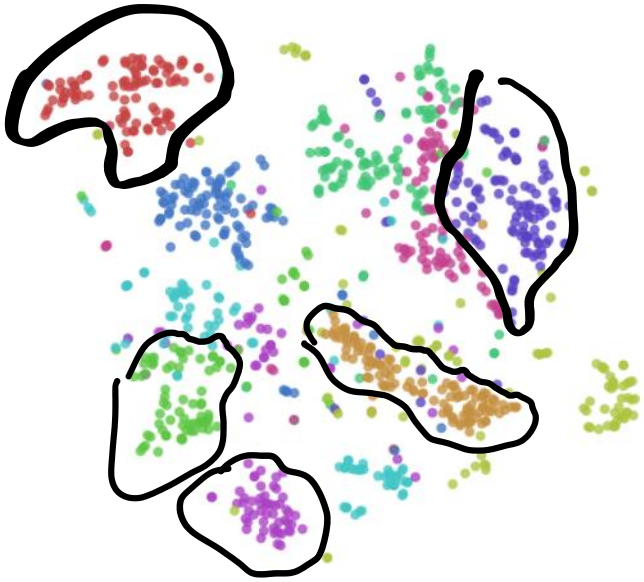
Furthermore, when you map the high dimensional data in two dimensions, the area used by the data points in two dimensions that have high distances in the high dimensional data will not be sufficiently larger than the area used by the data points in two dimensions that have low distances in the high dimensional data. This is due to the fact that size scales up exponentially as dimensions increase. Consider the volume of sphere in n dimensions. It grows at a power of the radius. So, a sphere in n-dimensions will occupy much more space than a sphere in 2 dimensions, even with the same radius. So, the data in two dimensions tend to be 'crowded' together.

Since we are comparing the probabilities of the data points and not the actual distances, one natural way to overcome this issue is to consider distributions with fatter tails than Gaussian. One such distribution is the Student-t distribution. This works because when the tails are fatter, the distance of the points in the lower dimensional space is much bigger compared to when the tails are normally distributed.

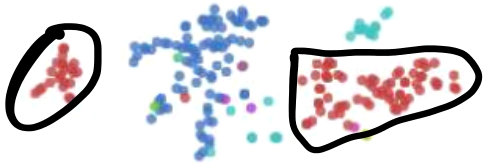
To visualize it, consider the following. The z value of a cumulative probability score in the Student-t distribution is more than the z value of the cumulative probability score in the Gaussian distribution. So, the formula for the probabilities in the lower dimensional space now reduces to:

$$q_{i|j} = \frac{(1+||y_i - y_j||)^{-1}}{\sum_{k \neq l}(1+||x_i - x_k||)^{-1}}$$

The optimization of the KL divergence is done using gradient descent algorithms that every data science enthusiast should be familiar with. However, there are some improvements made such as early exaggeration, which means the probabilities p are multiplied first early in the optimization, encouraging large values for p and fairly large values for q. This means that the space between clusters is maximized and is better for visualizations. Another improvement is adding the L2 regularization of the squared distances of the data, which promotes the clustering of data points.

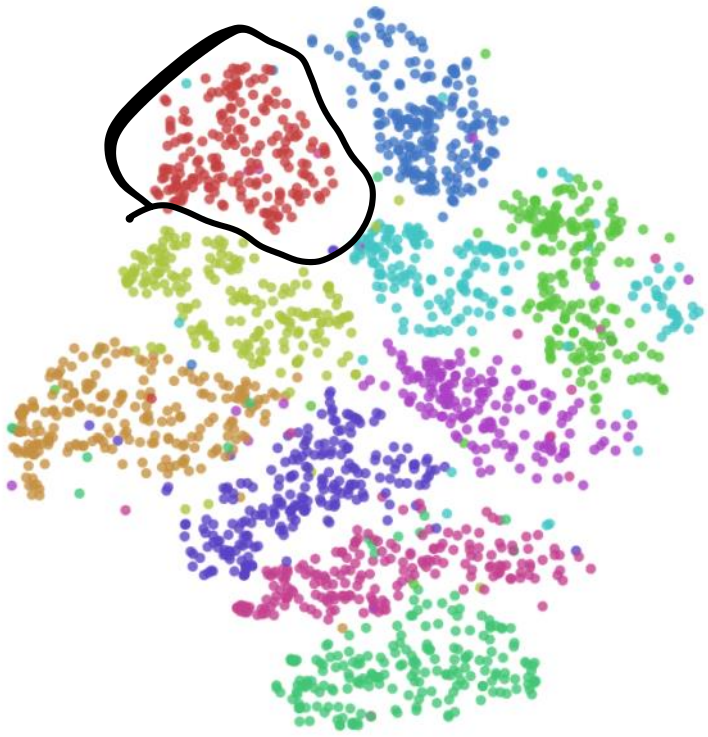


t-SNE does an impressive job finding clusters and subclusters in the data, but is prone to getting stuck in local minima. For example, in the following image we can see two clusters of zeros (red) that fail to come together because a cluster of sixes (blue) get stuck between them.



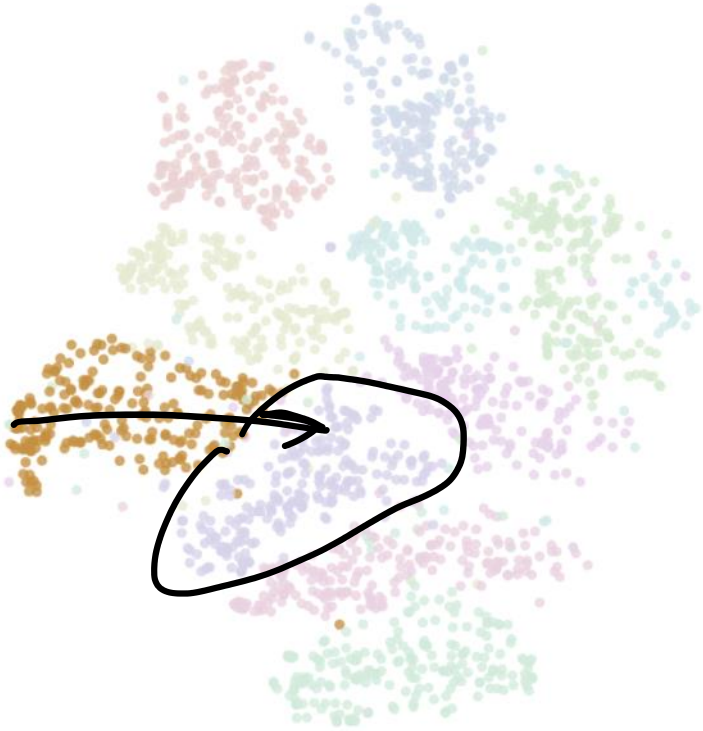
A number of tricks can help us avoid these bad local minima. Firstly, using more data helps a lot. Because these visualizations are embeded in a blog post, they only use 1,000 points. Using the full 50,000 MNIST points works a lot better. In addition, it is recommended that one use simulated annealing (http://en.wikipedia.org/wiki/Simulated_annealing) and carefully select a number of hyperparamters.

Well done t-SNE plots reveal many interesting features of MNIST.

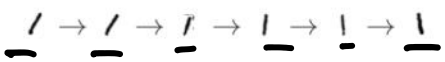


An even nicer plot can be found on the page labeled 2590, in the original t-SNE paper, Maaten & Hinton (2008) (http://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf).

It's not just the classes that t-SNE finds. Let's look more closely at the ones.



The ones cluster is stretched horizontally. As we look at digits from left to right, we see a consistent pattern.



They move from forward leaning ones, like /, into straighter like |, and finally to slightly backwards leaning ones, like \. It seems that in MNIST, the primary factor of variation in the ones is tilting. This is likely because MNIST normalizes digits in a number of ways, centering and scaling them. After that, the easiest way to be "far apart" is to rotate and not overlap very much.

Similar structure can be observed in other classes, if you look at the t-SNE plot again.

In three dimensions, MDS does a much better job separating the classes than it did with two dimensions.

And, of course, we can do t-SNE in three dimensions.

https://distill.pub/2016/misread-tsne/

3D Visualization

Visualization in Three Dimensions

Watching these visualizations, there's sometimes this sense that they're begging for another dimension. For example, watching the graph visualization optimize, one can see clusters slide over top of each other.

Really, we're trying to compress this extremely high-dimensional structure into two dimensions. It seems natural to think that there would be very big wins from adding an additional dimension. If nothing else, at least in three dimensions a line connecting two clusters doesn't divide the plane, precluding other connections between clusters.

In the following visualization, we construct a nearest neighbor graph of MNIST, as before, and optimize the same cost function. The only difference is that there are now three dimensions to lay it out in.

(plot)

In this visualization, we can begin to see why it is easy to achieve around 95% accuracy classifying MNIST digits, but quickly becomes harder after that. You can make a lot of ground classifying digits by chopping off the colored protrusions above, the clusters of each class sticking out. (This is more or less what a linear Support Vector Machine does.[4]) But there's some much harder entangled sections, especially in the middle, that are difficult to classify.

Of course, we could do any of the above techniques in 3D! Even something as simple as MDS is able to display quite a bit in 3D.

(plot)

Because t-SNE puts so much space between clusters, it benefits a lot less from the transition to three dimensions. It's still quite nice, though, and becomes much more so with more points.

If you want to visualize high dimensional data, there are, indeed, significant gains to doing it in three dimensions over two.

Conclusions on Dimensionality Reduction

Dimensionality reduction is a well developed area, and we're only scratching the surface here. There are hundreds of techniques and variants that are unmentioned here. I encourage you to explore!

It's easy to slip into a mind set of thinking one of these techniques is better than the others, but I think they're all complementary. There's no way to map high-dimensional data into low dimensions and preserve all the structure. So, an approach must make trade-offs, sacrificing one property to preserve another. PCA tries to preserve linear structure, MDS tries to preserve global geometry, and t-SNE tries to preserve topology (neighborhood structure).

These techniques give us a way to gain traction on understanding high-dimensional data. While directly trying to understand high-dimensional data with the human mind is all but hopeless, with these tools we can begin to make progress.