t-distributed Stochastic Neighbor Embedding (t-SNE)

We have already discussed a linear method namely PCA. Now we will discuss a non-linear method and will see what are the advantages of non-linear methods versus linear methods for visualizing data. And here we will see a newer method that really often performs very well for identifying clusters in our data set, which is known as t-SNE or t-distributed Stochastic Neighbor Embedding.

t-distributed Stochastic Neighbor Embedding (t-SNE)

- Probabilistic approach to place samples from high-dimensional space into low-dimensional space so as to preserve the identity of neighbors.
- Find an embedding so that original high-dimensional sample distribution is approximated well by the resulting low-dimensional sample distribution (t-SNE uses Kullback-Leibler divergence to measure the "distance" between distributions and minimizes this objective function)
- It gives rise to a non-linear embedding where close-by points remain close by and far away points remain far away so that clusters are preserved.

So what is the idea behind all this?
The idea is actually quite interesting. So we have our samples i.e customers that live where we have these high dimensional measurements, say the buying behavior over a long timeframe. So we have our samples in this high-dimensional space and that defines a sample distribution.
We would like to represent that distribution in a low dimensional space, maybe in a two-dimensional space so that we can actually visualize the space. So we can think of all of our customers in this two-dimensional space and that again will define a sample distribution.

The idea behind Stochastic Neighbor Embedding is to view our samples as a sample distribution in a high dimensional space and a distribution in a low dimensional space. And we would like these two distributions to be as similar as possible to each other. Now there are different ways of measuring distances between distributions.

One of them which is used quite a lot in information theory for example is known as Kullback-Leibler divergence. And that's exactly what is used in t-SNE for actually defining a distance between these two distributions -that's what we want to minimize, so that's our objective function.

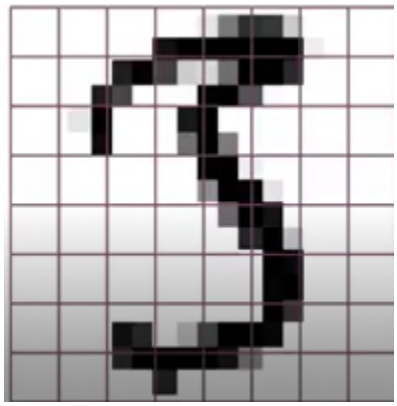
We want these two distributions to look as similar as possible. So what this method does is, it tries to minimize the differences between these two distributions, and that's quite a difficult optimization problem. We won't go into the intricacies of solving this optimization problem, but it's highly non-convex, meaning that it will depend on the initialization that we're giving it. So we should give a good initialization often when we actually use this PCA embedding, which we discussed before. It tries to find the good starting embedding and then from there it optimizes more and more for this difference between the two distributions to become smaller and smaller. So it's a difficult optimization problem and computationally this method is quite heavy. But we'll see some examples showing that it's really often worth doing something like this i.e something nonlinear to find a good embedding where we can really actually start to see clusters in our dataset. So, non-linearity helps us to do it.

We want the two distributions to still look similar to each other meaning where we have a high density of our samples, they should remain high density of samples. So that the clusters that we have remain exactly there i.e we keep the clusters preserved. And clusters that are further away from each other i.e for separated clusters we want them to still be separated in a low dimensional embedding and that's the goal. We preserve clusters and clusters that are separated from each other, we would like to preserve that as well, and that's possible with these nonlinear methods like stochastic neighbor embedding. It's very hard to do this actually with linear embeddings because if we just want to do a projection of high dimensional space, like a 10,000-dimensional space into a two-dimensional space, then that means that in 10,000-dimensional space there's so much more space than in a two-dimensional space. So, the problem of just doing something linear is that, if we're going from something with a lot of space to something with little space then there will be crowding. So usually what happens is that all of the points, even though in this high dimensional space they might still be kind of separated into different clusters, they now all become one cluster. So that's always a difficulty when we just use linear methods. Nonlinear methods like stochastic

neighbor embedding, allow us the added non-linearity to still spread out the data better than just a linear approach. And we'll see what actually happens i.e why and how this method can actually help us quite a bit as compared to PCA which is a linear method.

Let's go through an example. So here's a very simple dataset that is actually very often used for showing or demonstrating different kinds of approaches in machine learning. This is a digit recognition dataset called MNIST, and we have a whole lot of hand-written digits.

- MNIST contains about 1800 images of hand-written digits - 180 for each digit from 0 to 9.
- Each (centered) digit was put in an 8 * 8 grid (i.e D = 64)
- Measure the gray value in each part of the grid, i.e 64 gray values
- Input : x_1 ,..., $x_n \in R^D$ Output: y_1 ,..., $y_n \in R^d$, where d << D

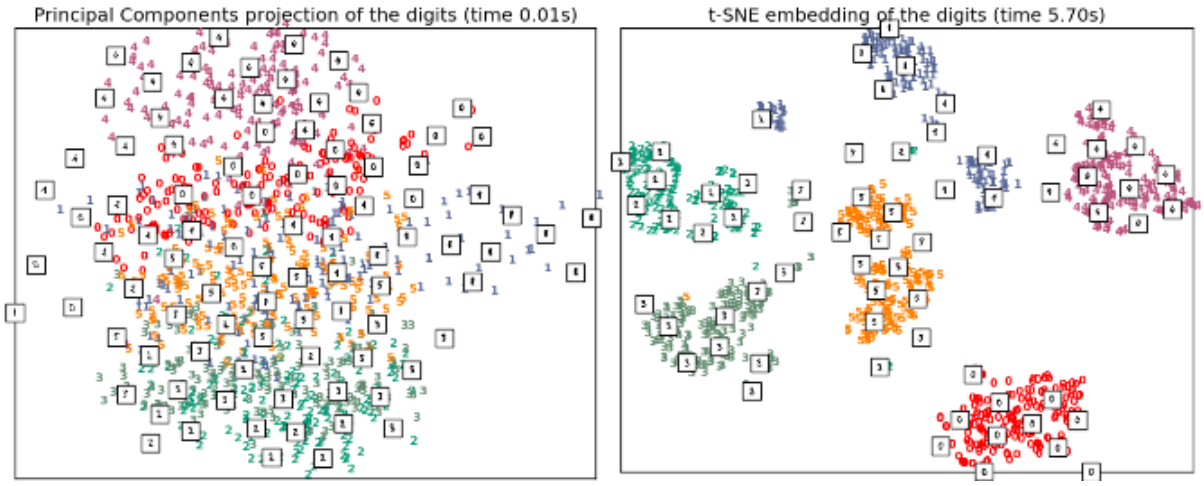


So this is how these hand-written digits look. Every sample here is one hand-written digit. They are summarized into a 64-dimensional vector and it's in grayscale format, so each one of these pixels has a value assigned to it for that kind of entry.

Now we would like to represent all these digits in a low dimensional space say two-dimensional space, so that we can actually look at it and we want to see what are the differences in the representation if we use PCA to represent our data as compared

to this nonlinear method, t-SNE, which just tries to somehow make the two sample distributions in the high dimensional space and low dimensional space as similar as possible.

So let's see what this looks like and look at some of the differences and similarities.



We choose digits from 0-5 to do this.
So this is what we get with PCA, we have the first principal component and similarly the second principal component. And, as we discussed, the first Eigenvector corresponds to the largest Eigenvalue and the second Eigenvector corresponds to the second largest Eigenvalue. This process is super fast (taking only 0.01 second) i.e doing a PCA plot is really really fast.

And the stochastic neighbor embedding is much slower. It took 5.7 seconds to get these embeddings here. Also, this depends on our initialization. So if we would run this again with a different initialization, we might actually get a different kind of picture.
So we really want to run this a couple of times, and then just choose an embedding that leads us to the smallest objective function. The objective function is the Kullback-Leibler divergence that the algorithm will actually output to us as well. So we can just use the results that have the smallest value of this Kullback-Leibler divergence.

So what do we see here as differences?
It's quite clear when we compare the above two pictures against each other.
For PCA we have one big blob, and for t-SNE what we see is actually quite nice, it seems to find meaningful clusters. So without telling the algorithm anything about that we have different numbers, and different digits, we just took the 64-dimensional vectors and all we did is just visualize it in two dimensions. It's actually able to identify that there are differences between these different digits. Here, all the 2's are clustered together, all the 3's are clustered together and similarly, all other digits are clustered together. So that's quite nice.

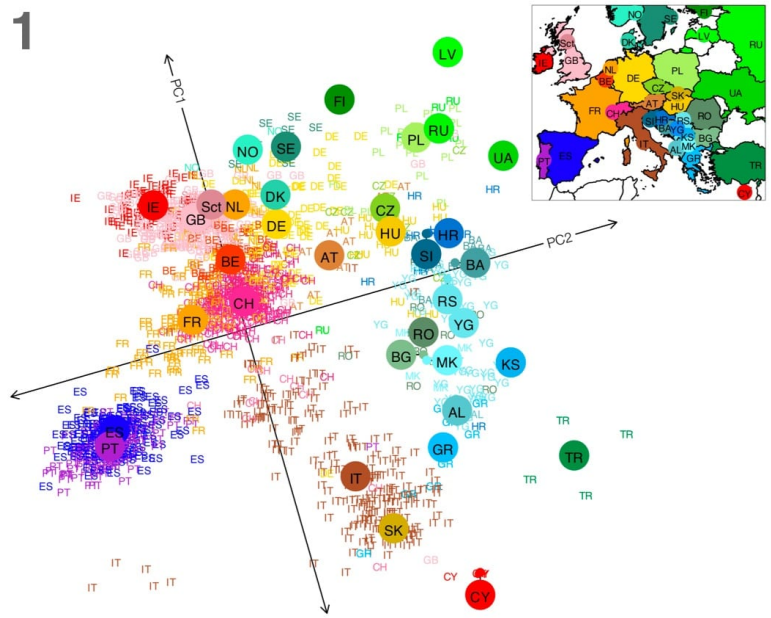
Now we can imagine that if we do this, using a large customer base, that might be for example the whole buying history from these customers. Then a plot i.e a t-SNE visualization like this will be able to group out and separate customers into very different segments or different groups. And then we can go in and actually try to analyze what they actually mean.

Whereas for PCA it's quite difficult to see. We see that along with the first principal component, we have mainly the number 1 that is really spread out. So that is the number with the largest variation because it goes all along with the first principal component, and then the second principal component was kind of able to distinguish between different digits and all the digits are separated along with the second principal component, but it's still not clear where different clusters are. As, because of this linear embedding, we're going from 64-dimensional space to two-dimensional space and there's so much less space that we have to expect that we basically get one blob, although maybe in the high dimensional spaces we did actually have multiple blobs.

So if our goal is just to identify outliers, for example, if there was one number that was completely an outlier. We actually often identify that quite easily in PCA, where we have some samples that are very far out. PCA is a very fast way of doing that, so it is very interesting from that perspective. So definitely, it is advisable to do PCA as a first step so that you can maybe see some patterns that you can just get out very quickly.

But if you want to start seeing different groups in your dataset, then you can go for something like t-SNE which takes longer but it really identifies something interesting in your data. It might also see patterns, and then you can go in and try to figure out why all these customers are grouped together? What is it that makes them similar to each other and different from the customers that you have in other groups?
So we discussed the advantages and disadvantages of these two methods.

Of course, we saw that for some applications, PCA can already work very well, such as the application that we had seen here at the beginning.



Here we also saw that some clusters still come out and are separated from each other, but it was basically put into one big blob in the hand-written digits example.
However the location still made a lot of sense.

But for most other applications, we may need something that is a bit more complex on the computational side, like t-SNE, which takes longer as it solves a more difficult optimization problem. But then it can actually identify some really interesting structure in the data just through a simple visualization.