t-Distributed Stochastic Neighbor Embedding

D2K Course Staff

2/21/2019

t-Distributed Stochastic Neighbor Embedding (t-SNE)

A more modern method that has gained recent popularity as a dimension reduction technique is t-SNE. Compared to PCA, it often performs better in identifying clusters and is more helpful in exploratory analysis since it weighs data points that are closer together more heavily than linear methods. The key distinction between t-SNE and linear methods is that it takes the Euclidean distance in high-dimensional space, and then maps it to a conditional normal probability function to represent the distances, or similarities.

We can define the conditional probability $p_{j|i}$ as the similarity of data point x_j to x_i, or the probability that x_i would pick x_j as its neighbor. This probability is determined by a normal distribution which means that for nearby data points, the similarities are very high, but for highly separated data points this probability becomes extremely small. We can represent the conditional probability as,

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\Sigma_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

where σ_i^2 is the variance of the normal distribution defined for x_i.

We can apply a similar logic to our lower-dimensional features to define a conditional probability metric for similarity between the lower-dimensional data points. However, we define the similarity metric using the Student's t-distribution with one degree of freedom as it allows us to model a moderate distance in the original data to a much larger distance in the lower-dimensional mapping. This allows us to eliminate the attractive forces between moderately dissimilar data points and thus achieve better separation for pattern recognition. We can represent this joint probability as,

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\Sigma_{k \neq i}(1 + ||y_i - y_k||^2)^{-1}}$$

where we there is only one degree of freedom.

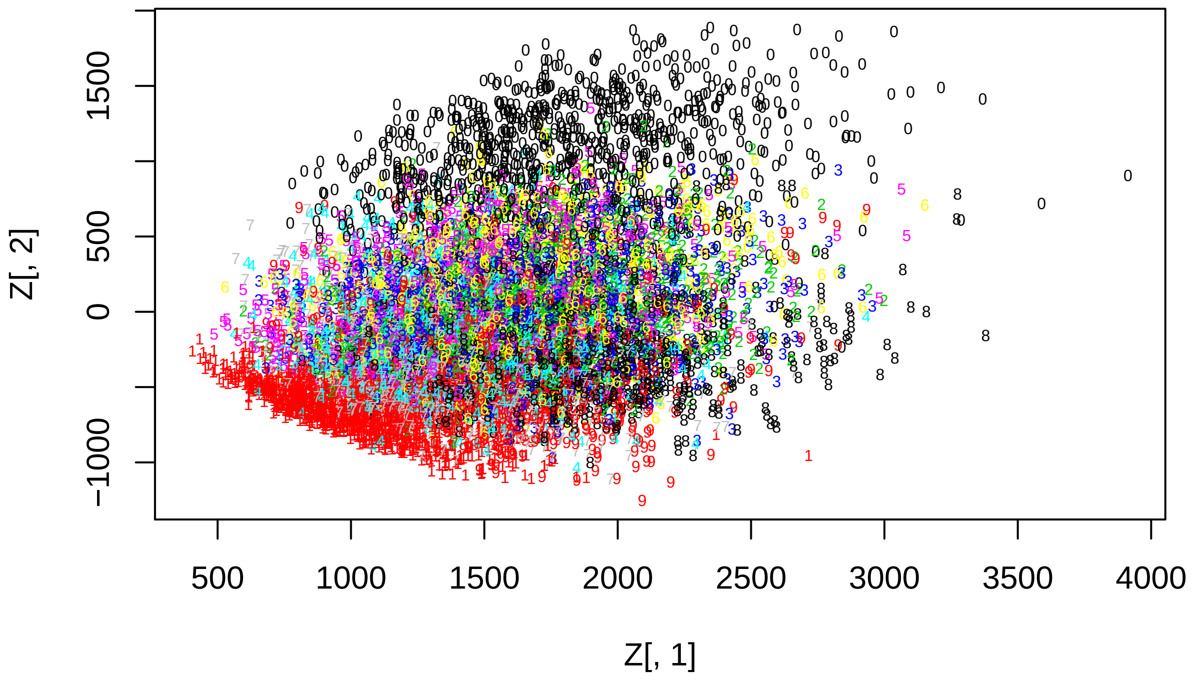
We also define the cost function, or the objective equation we are seeking to minimize, is the sum of the Kullback-Liebler divergences. Mathematically, it can be represented as,

$$C = KL(P||Q) = \Sigma_i \Sigma_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

where we define the joint probability p_{ij} as the symmetrized conditional probabilities, $p_{ij} = \frac{p_{j|i} + p_{ij}}{2n}$. We solve this optimization problem using the gradient descent algorithm. Using normal distributions in the higher-dimensional space, and Student's t-distributions in the lower-dimensional space allows us to avoid the "crowding" problem that occurs in techniques like MDS.

We will now show an implementation of t-SNE in R. We use a dataset filled with pixel values to highlight how t-SNE performs better than techniques like PCA for visualization. First we show how PCA performs on the data set.

```r
# Load t-SNE library
library(Rtsne)

# Load data
pixels <- read.csv("data/train.csv", header = TRUE, stringsAsFactors = FALSE)

# Create data matrix of numeric values
pixels_x <- as.matrix(pixels[,-1])

# Create vector of labels
pixel_labels <- as.factor(pixels[,1])

# Set seed for reproducibility
set.seed(123)

# PCA using SVD
X <- pixels_x
rownames(X) <- pixel_labels
sv <- svd(X);
V <- sv$v
Z <- X%*%V;

# Visualize PCA
par(mfrow=c(1,1))
plot(Z[,1],Z[,2],type="n")
text(Z[,1],Z[,2],rownames(X),col=as.numeric(pixel_labels),cex=.5)
```

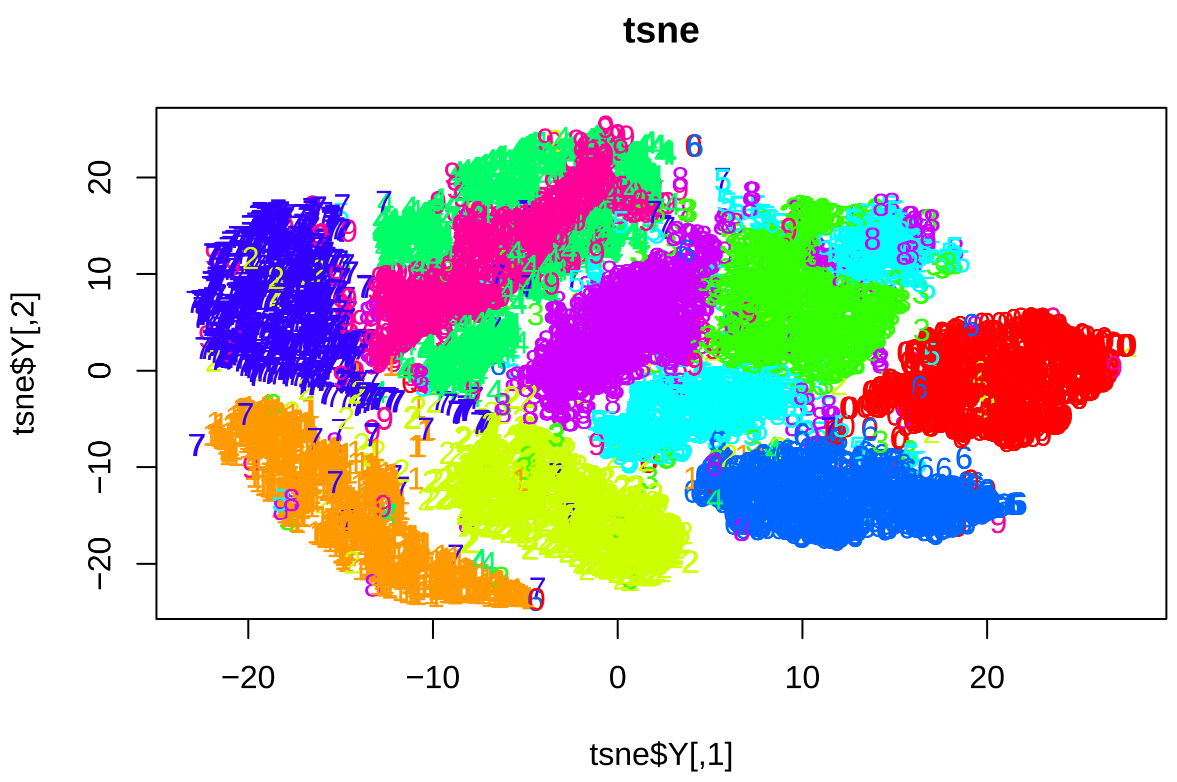


As you can see, some patterns start to arise. However, we clearly have a "crowding" since clusters of data are not clearly separated and there is much overlap between the clusters. We can see if t-SNE performs any better on the data and if it avoids the "crowding" problem.

```r
# t-SNE
tsne <- Rtsne(pixels_x, dims = 2, perplexity=30, verbose=TRUE, max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 12.82 seconds (sparsity = 0.012259)!
## Learning embedding...
## Iteration 50: error is 97.574505 (50 iterations in 2.67 seconds)
## Iteration 100: error is 90.572766 (50 iterations in 4.73 seconds)
## Iteration 150: error is 86.514037 (50 iterations in 2.67 seconds)
## Iteration 200: error is 86.201899 (50 iterations in 2.59 seconds)
## Iteration 250: error is 86.160226 (50 iterations in 2.75 seconds)
## Iteration 300: error is 3.140585 (50 iterations in 2.47 seconds)
## Iteration 350: error is 2.744775 (50 iterations in 2.14 seconds)
## Iteration 400: error is 2.528705 (50 iterations in 2.15 seconds)
## Iteration 450: error is 2.386856 (50 iterations in 2.35 seconds)
## Iteration 500: error is 2.284919 (50 iterations in 2.20 seconds)
## Fitting performed in 26.73 seconds.
```

```r
# Create color palette for visualization
colors <- rainbow(length(unique(pixel_labels)))
names(colors) <- unique(pixel_labels)

## Plot t-SNE
par(mfrow=c(1,1))
plot(tsne$Y, t='n', main="tsne")
text(tsne$Y, labels=pixel_labels, col=colors[pixel_labels])
```



Look how well t-SNE separates the data into distinct clusters compared to PCA! Generally, it is best practice to try both PCA and t-SNE, along with other methods, when trying to perform dimension reduction techniques for visualizaiton and exploration.

Advantages

- Better at distinguishing patterns and clusters than PCA
- Works better to find relationships that are *non-linear*
- Avoids the "crowding" problem

Disadvantages

- Results can depend on the random starting values of Y_i chosen
- Computationally expensive (infeasible for data sets with n >> 10,000)
- The method does not actually assign points to clusters, in the same manner that your basic k-means or your convex clustering methods would. The idea here is just to find the low-dimenisonal representation that best maintains relative pairwise distances between the points. If we want to get automatically assigned clusters, we could then run a clustering algorithm on the low-dimensional representation returned by t-SNE.

References

Saurabh, and University of California. "Comprehensive Guide on t-SNE Algorithm with Implementation in R & Python." Analytics Vidhya, 22 Mar. 2017, www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/.

Van der Maaten, Laurens, and Geoffrey Hinton. “Visualizing Data Using t-SNE.” Journal of Machine Learning Research, vol. 9, Nov. 2008, pp. 2579–2605., lvdmaaten.github.io/publications/papers/JMLR_2008.pdf.