Software engineering module 1(Python)

High dimensional data, dimensionality reduction, and manifold learning

Jian Zhou

Overview of Module 1

- **Mathematical properties** of high dimensional space
- Overview of dimensional reduction methods
- **Implement t-SNE algorithm**
- **Implement GraphDR algorithm**
- Elements of scientific **computing performance**:
  - *Computational complexity* of common operations
  - Ideas for designing *faster algorithm (*Multiscale coarse-graining, FFT, randomization)
  - *Implementation* (GPU kernel, BLAS, MKL, Cython, etc)
- **Python code and documentation style. Auto-generation of documentation website with Sphinx**
- **Anatomy of python scientific software (API, CLI, README, Docs, LICENSE).**
- **Basics of Git, Github, pull request, & code review**
- Demo

Grading for module 1 (20% of total grade)

Plan A:

No ChatGPT (or other AI tools)

Code 12% (correctness, efficiency, readability)
Documentation 5%
Code review 3%
Exploratory tasks *0-5% (optional tasks or additional features / explorations)

*You can receive bonus points from exploratory tasks, but total grade won't exceed 20%

Plan B:

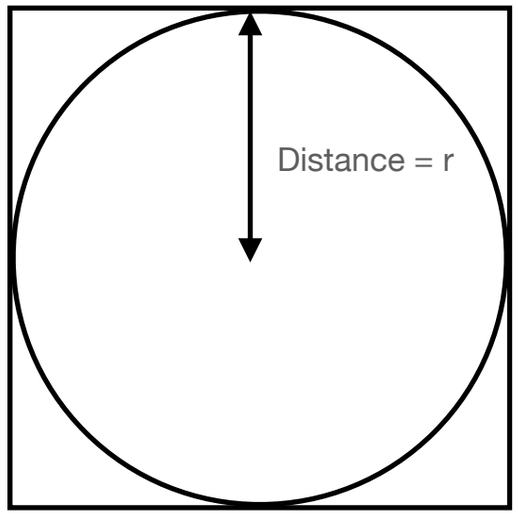
ChatGPT or AI allowed

Code + Documentation + Code review 10% (Pass or fail)
Exploratory tasks 10% (optional tasks or additional features / explorations)

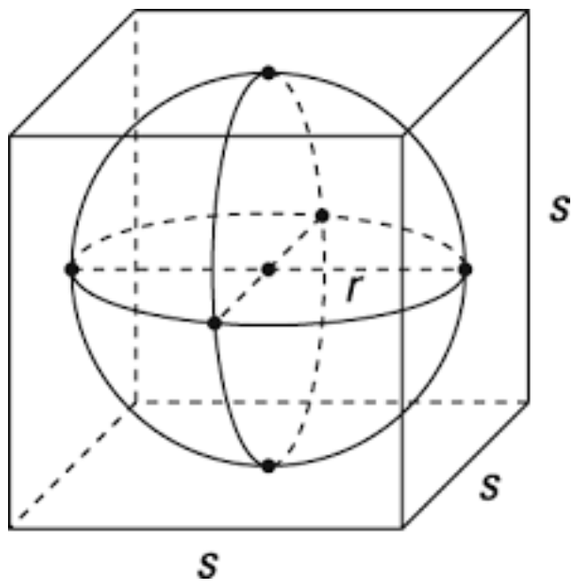
Exploratory tasks are scored only based on your own ideas, if the idea comes from someone else, it must be explained in the presentation and won't add to your score

Properties of high dimensional space

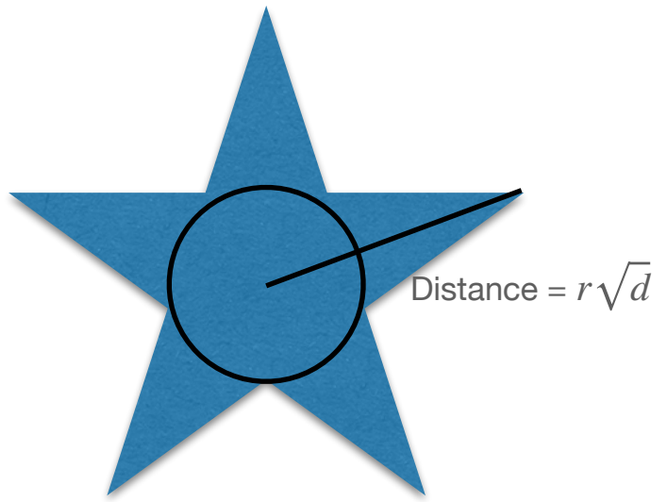
d=2



d=3



high d



$$\frac{V_{\text{hypersphere}}}{V_{\text{hypercube}}} = \frac{(d/2)!}{\pi^{d/2}} = \frac{1}{d^{\Theta(d)}}$$

- For KNN classifier, proportion of space within a unit radius hypersphere is increasingly small in high dimensions
- The proportion of space with high probability under an isotropic Gaussian distribution is increasing small in high dimensions

Properties of high dimensional space

Number of almost orthogonal vectors that fit into d-dimensional space

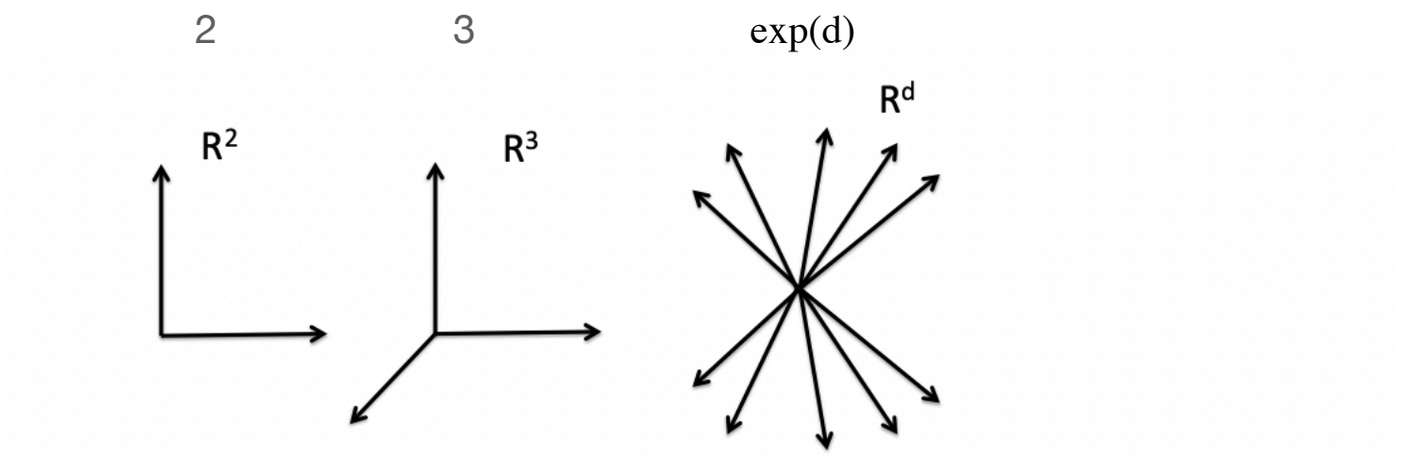


Figure 1: Number of almost-orthogonal vectors in $\Re^2, \Re^3, \Re^d$

Random vectors are *almost* orthogonal to each other

$$\cos(\theta) = \frac{\langle x, y \rangle}{\|x\|_2 \|y\|_2}$$

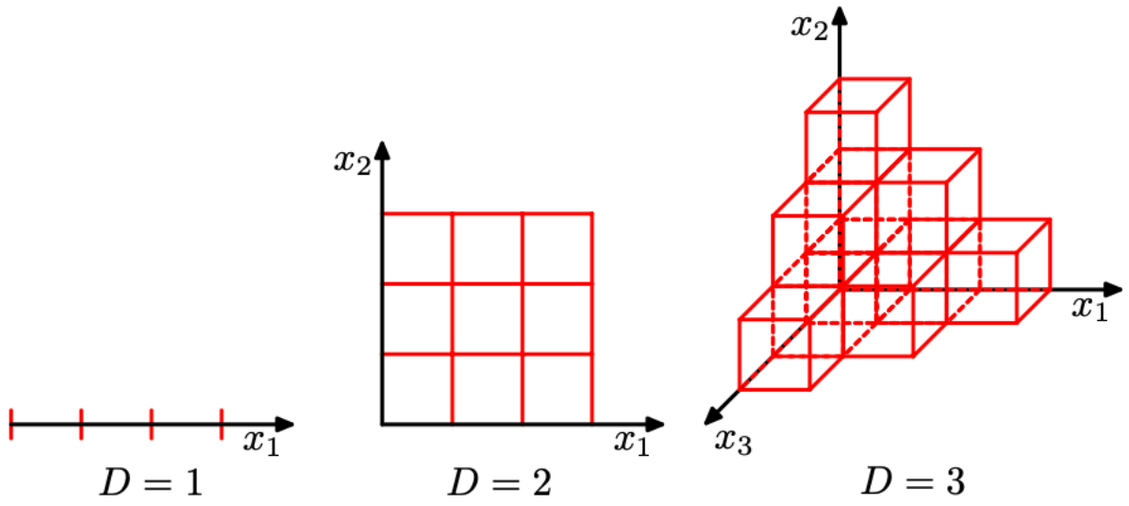
```
[-1, -1,  1,  1,  1,  1,  1,  1,  1,  1, …]
[ 1,  1, -1,  1,  1,  1, -1,  1, -1,  1, …]
```

Proof sketch: With high probability, random vectors are almost orthogonal (exponential with d) with Chernoff bound,
With union bound, the number of vectors that are almost orthogonal with high probability grows exponentially with d too.

The “Curse of dimensionality”

The term comes from Bellman, 1961, on combinatorial optimization problems and used to motivate dynamic programming.

It is now widely used to refer to the phenomenon in many field that difficulties arise with high number of dimensions.



Pattern Recognition and Machine Learning, Christopher Bishop 2006 Chapter 1.4

High dimensional data has desirable properties too: so-called “blessings of dimensionality”

Real data is typically confined in a region with low effective dimensionality

Johnson–Lindenstrauss lemma

Given a set X of m points in $\mathbb{R}^N$, we would like to find a mapping $f\,\mathbb{R}^N \to \mathbb{R}^n$ where n << N, so that

$$(1-\varepsilon)\|u-v\|^2 \leq \|f(u)-f(v)\|^2 \leq (1+\varepsilon)\|u-v\|^2 \quad \text{, for any } u, v \in X$$

In other words, the distance between all pairs of points in preserved.

Johnson–Lindenstrauss lemma: $n = \mathcal{O}\left(\frac{\log m}{\epsilon^2}\right)$ dimensions is sufficient, and the mapping can be a linear mapping.

Sketch of proof: consider **random projection (e.g. random sign, random Gaussian projection)** from $\mathbb{R}^N$, such that the **expectation** of $\|f(u)-f(v)\|^2$ equals to $\|u-v\|^2$, then show that the value of $\|f(u)-f(v)\|^2$ **is close to** $\|u-v\|^2$ with sufficiently high probability

For proof, find https://www.cs.princeton.edu/courses/archive/fall16/cos521/Lectures/lec9.pdf

Remark 1: this applies to any data in $\mathbb{R}^N$. In other words - this is the worst case scenario of any data
Remark 2: the reduced dimensionality depend on number of data points m but not original dimensionality N
Remark 3: this proof only applies to L2 distance, and it has been proven that L1 distance cannot be preserved

Improve over Random Projection with data-dependent linear projection

Problem 1:

Consider low dimensional embedding Z of the high-dimension data X, so that

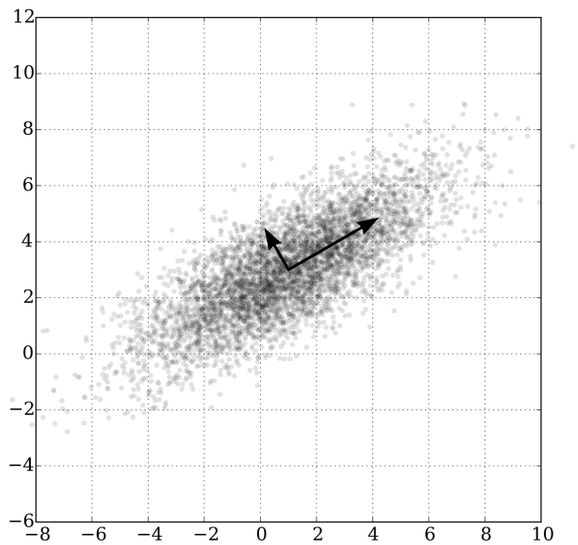
$$\underset{Z}{\text{minimize}} \sum_{ij} \Big(\|X_i - X_j\| - \|Z_i - Z_j\| \Big)^2$$

Problem 2:

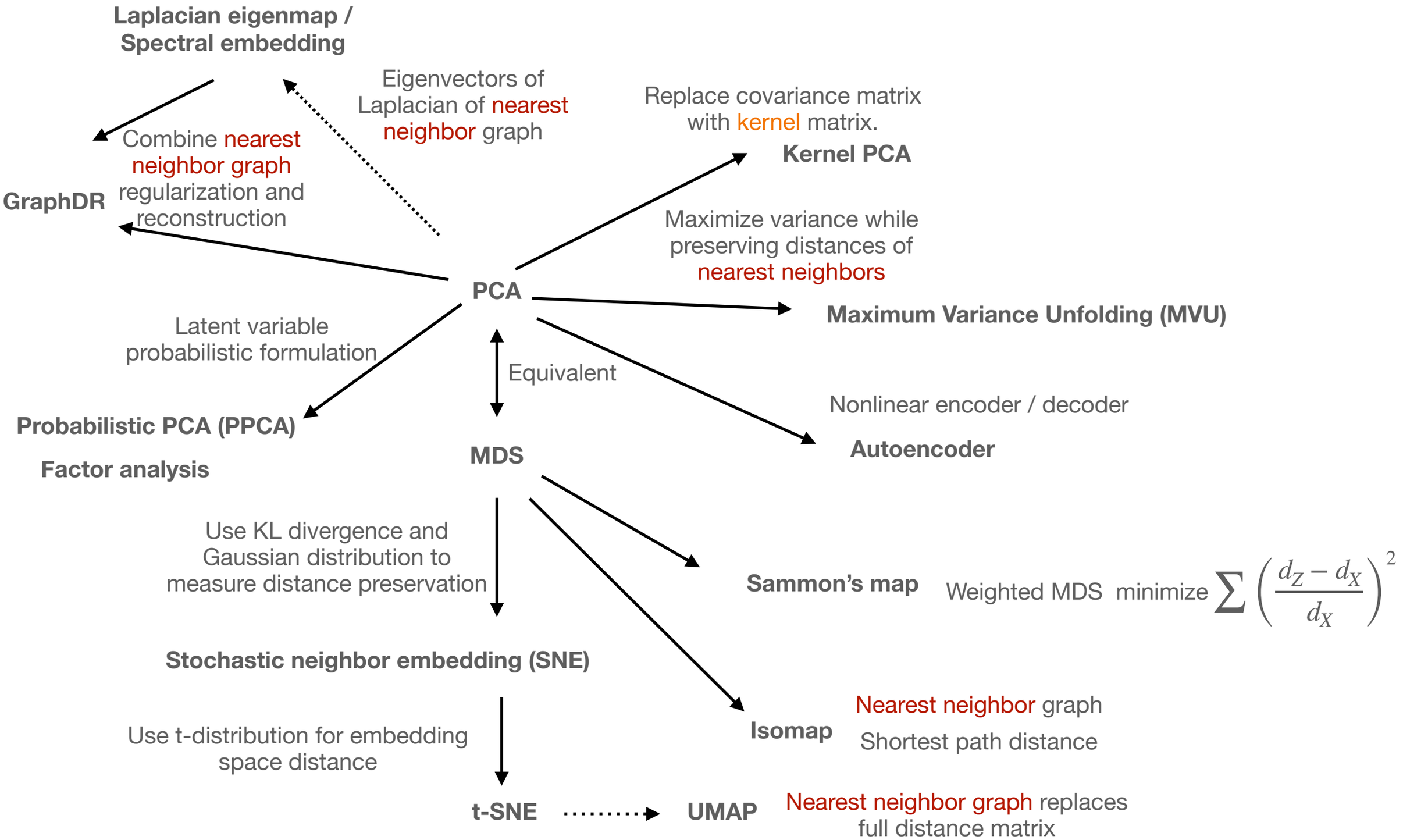
Consider low dimensional embedding Z of the high-dimension data X, which we express as $Z = XW$,

$$\underset{W}{\text{minimize}} \quad \|X - ZW^T\|^2 \quad \text{subject to} \quad W^T W = I$$

Both formulations (often called classical multidimensional scaling or principal component analysis respectively) are equivalent and lead to the same solution:

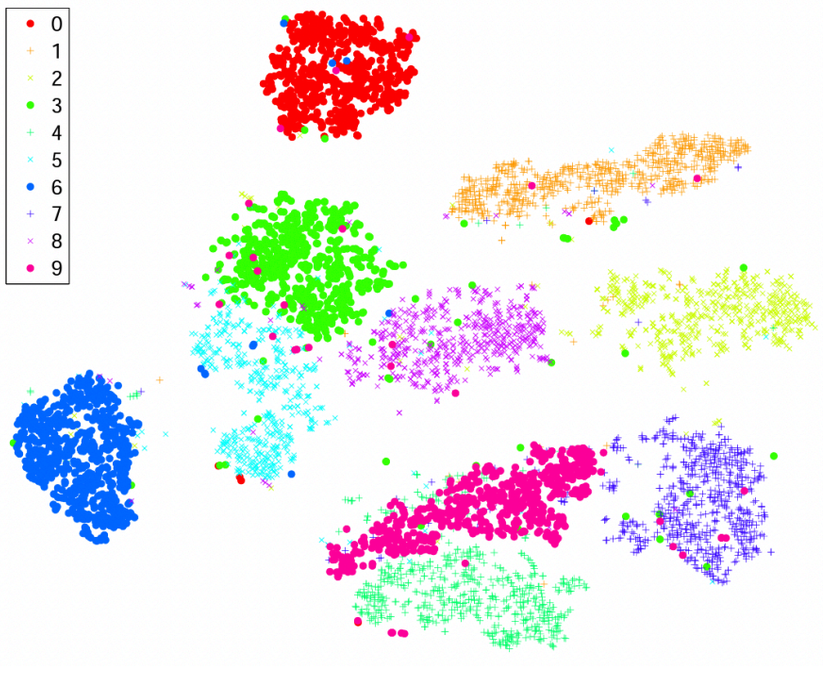


A map of dimensionality reduction methods*

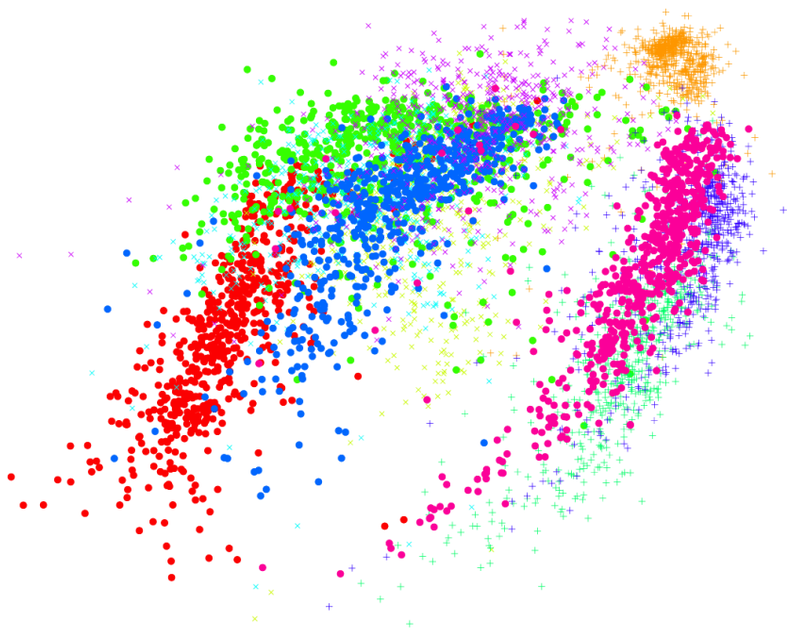


* Note there is more than one way to draw connections between methods and often more than one way to interpret a method

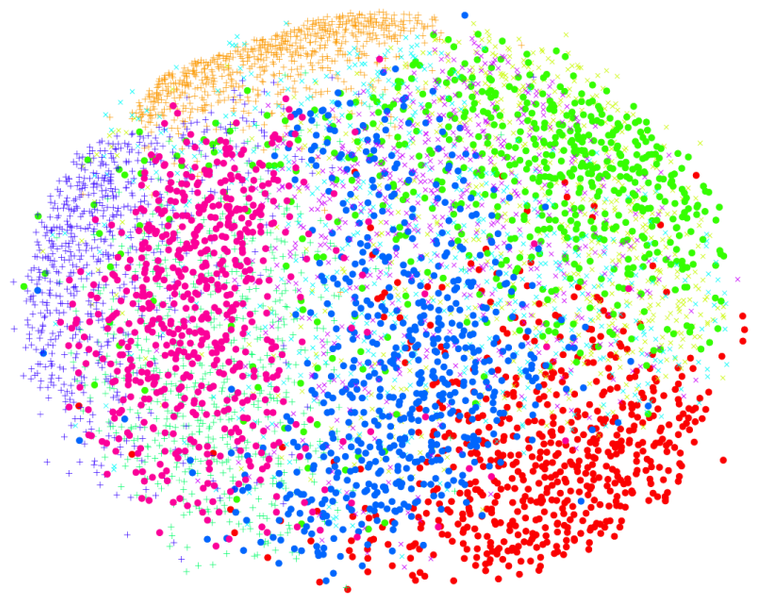
Comparison of t-SNE, Isomap and Sammon’s map



t-SNE



Isomap



Sammon’s map

t-SNE (t-distributed stochastic neighbor embedding)

Input space X N x d

Represent distance by Gaussian

$$p_{j|i} = \frac{exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2/2\sigma_i^2)}, \; p_{ij} = \frac{p_{j|i} + p_{i|j}}{\sum_{\substack{k,l \\ l \neq k}} (p_{k|l} + p_{l|k})}$$

Objective: Minimize KL Divergence of P and Q

$$\mathrm{KL}(P \parallel Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Embedding space Y N x 2

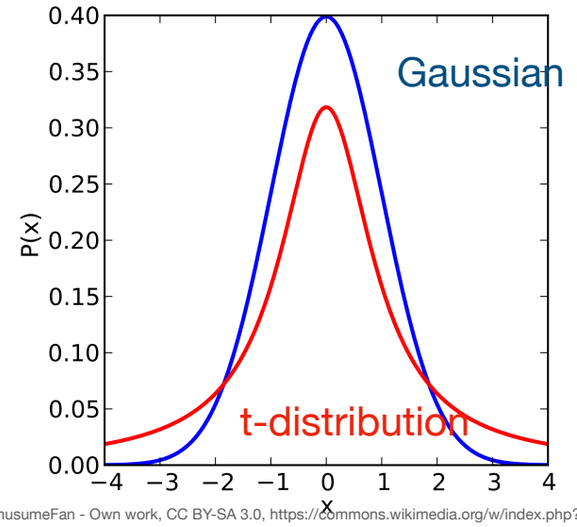
Represent distance by t-distribution (df=1)

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

Gradient

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1}$$

Comparison of Gaussian and t-distribution



We will begin by rewriting the gradient update rule in the following form

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \left(\underbrace{\sum_{j \neq i} p_{ij} q_{ij} Z (\mathbf{y}_i - \mathbf{y}_j)}_{\text{Attractive force}} - \underbrace{\sum_{j \neq i} q_{ij}^2 Z (\mathbf{y}_i - \mathbf{y}_j)}_{\text{Repulsive force}} \right)$$

where Z is defined as the normalization constant in q_{ij}

$$Z = \sum_{k \neq l} \left(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2\right)^{-1}$$

But there's more: t-SNE algorithm breakdown

Loss function:

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

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1}$$

Optimization:

Gradient descent

$$Y_t = Y_{t-1} + \text{step size} \cdot g_t$$

Gradient descent+momentum

$$Y_t = Y_{t-1} + \Delta Y_t$$

$$\Delta Y_t = \text{momentum} \cdot \Delta Y_{t-1} + \text{step size} \cdot g_t$$

Gradient descent+momentum+ dynamically adjust step size

Increase step size — If momentum and g signs agrees

Decrease step size — If momentum and g signs disagrees

Early Exaggeration

Multiply p_{ij} by a factor in early iterations,
increase attractive / repulsive force ratio to allow points to move more freely in early iterations

t-SNE implementation: choosing σ of Gaussian distribution

For each data point, choose a σ so that for the probability distribution $p_{j|i}$ computed from the distances has

entropy $$H = -\sum_j p_{j|i} \log p_{j|i}$$ $\approx$ log(Perplexity)

Why is perplexity considered the size of neighborhood?

We provide the code for choosing $\beta = \frac{1}{2\sigma^2}$

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

t-SNE algorithm pseudocode

Algorithm 1 t-SNE

Input: Data Array $X^{(n \times d)}$
Parameters: no_dims=2, perplexity=30.0, initial_momentum=0.5, final_momentum=0.8, η=500, min_gain=0.01, T=1000
Output: low-dimensional data representation Array $Y^{(n \times no\_dims)}$

Precision(β) adjustment based on perplexity (algorithm 2 and code)
Compute pairwise affinities p_{ij} (equation 1 and note 5.1)
Early exaggerate (multiply) $p_{ij}^{(n \times n)}$ by 4 and clip the value to be at least 10^{-12}
Initialize low-dimensional data representation Array $Y^{(0)}$ using first no_dims of PCs from PCA (code)
Initialize $\Delta Y^{(n \times no\_dims)} = 0$, gains$^{(n \times no_dims)} = 1$
for $t = 1$ **to** T **do**
  Compute low-dimensional affinities q_{ij} (equation 2 and note 5.2) and clip the value to be at least 10^{-12}
  Compute gradient dY (equation 3 and note 5.3)
  if t < 20 **then**
    momentum = initial_momentum
  else
    momentum = final_momentum
  end if
  Determine gains based on the sign of dY and ΔY :
  gains $= (\text{gains} + 0.2) * ((dY > 0) \neq (\Delta Y > 0)) + (\text{gains} * 0.8) * ((dY > 0.) == (\Delta Y > 0.))$
  Clip gains to be at least min_gain
  $\Delta Y = \text{momentum} * \Delta Y - \eta * (\text{gains} * dY)$
  $Y = Y + \Delta Y$
  if t $== 100$ **then**
    Remove early exaggeration via dividing $p_{ij}^{(n \times n)}$ by 4
  end if
end for

Peformance tip: use numpy broadcasting and array operation whenever you can!

Task for Day 1:

1. **Implement t-SNE following provided pseudocode and guide**
2. **Verify that it is working appropriately on the given MNIST example dataset** (code for loading the dataset and preprocess by PCA provided)
3. Optional: how well does t-SNE work for 3D embedding and can you improve it?

You may work together and help each other, but everyone has to write their own code. You can ask questions on teams in the Module 1 channel.

Tips for implementation and debugging

Break it down into parts, check your code as you implement part by part

Inspect your intermediate variables (array shape and values)

Use a test dataset (here we use MNIST as in the initial t-SNE paper)

Use an interactive computing environment like Jupyter notebook / Colab or/and an IDE (note that you need to write a .py module file not notebook)