Dimension Reduction Techniques for Data Visualization

Maria Cristina Ferreira de Oliveira
SCC5836/SCC0252

Data is…

Far too complex... (many dimensions, many types)

Far too big... (`easy´ to collect)

Multiple sources... (images, videos, documents, news feeds, sites, networks)

Never ending... (data streams)

Much redundancy...

Many relationships...

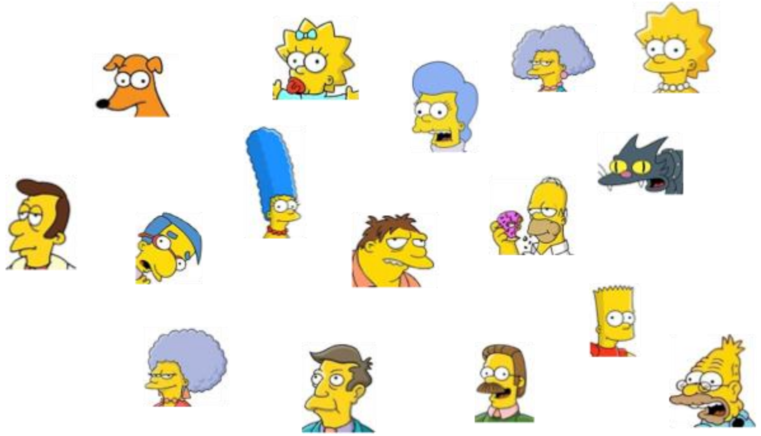
Missing pieces ...

Studying natural & artificial systems and phenomena implies in collecting & handling lots of high-dimensional data...

Visualization Problem

People try to make sense of data, more difficult if very high-dimensional

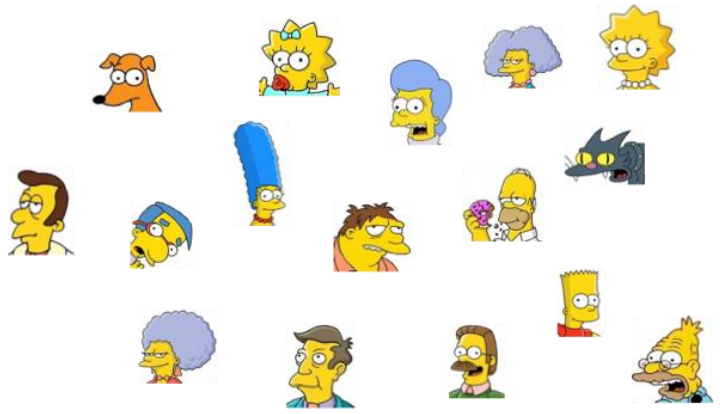
`noisy´ data

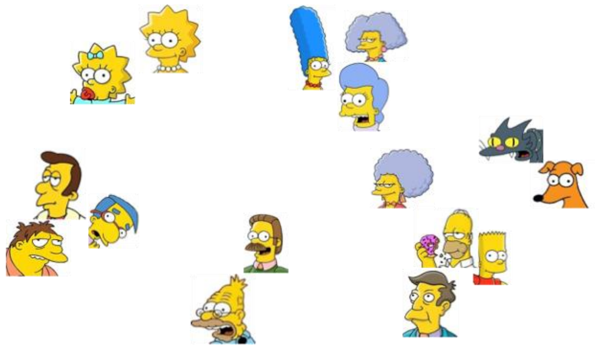


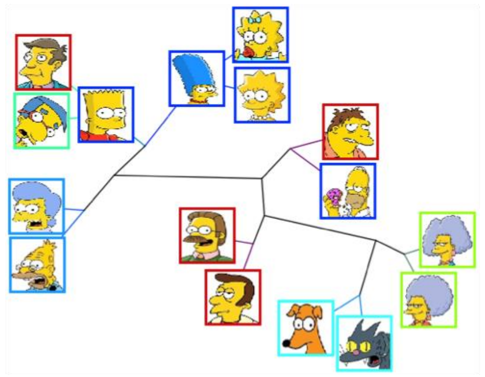
What can we tell about the data???

Are there relevant patterns?

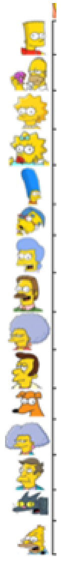
`noisy´ data







Data representation



5	12	15	2	7	5	0	12	9	0	8
12	5	0	12	12	12	12	12	18	12	12
0	1	05	10	15	12	8	12	9	11	5
0	12	01	12	9	0	12	10	5	5	12
12	8	05	12	12	12	8	12	9	12	12
10	12	0	11	10	2	7	12	2	16	7
5	6	8	12	12	15	12	6	9	17	0
7	12	05	0	12	12	10	17	9	12	12
2	10	05	15	12	1	12	10	9	8	2
12	12	7	12	0	12	0	12	10	12	12
6	12	05	17	12	10	12	12	9	12	8
12	10	2	12	1	12	12	11	6	0	12
1	12	05	12	12	16	2	12	9	12	0
10	0	12	12	9	12	0	10	12	12	8
0	12	1	12	12	5	1	7	11	12	12
8	2	11	10	7	12	5	12	15	10	0

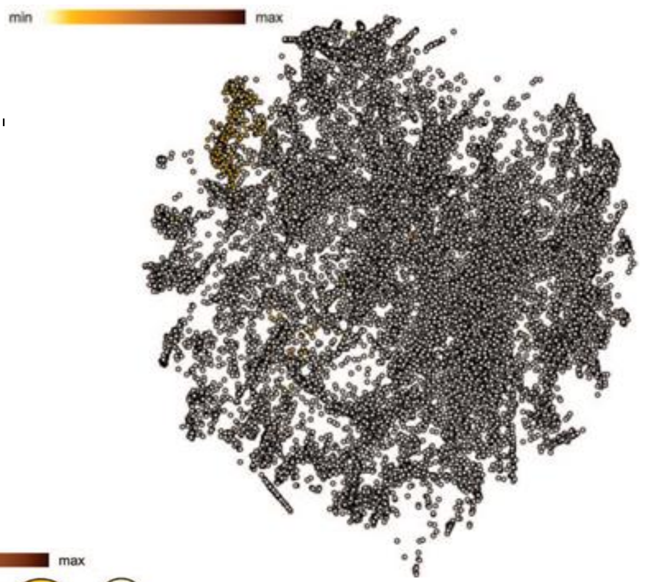
dimensional embedding
(data instance as a
vector of attributes)

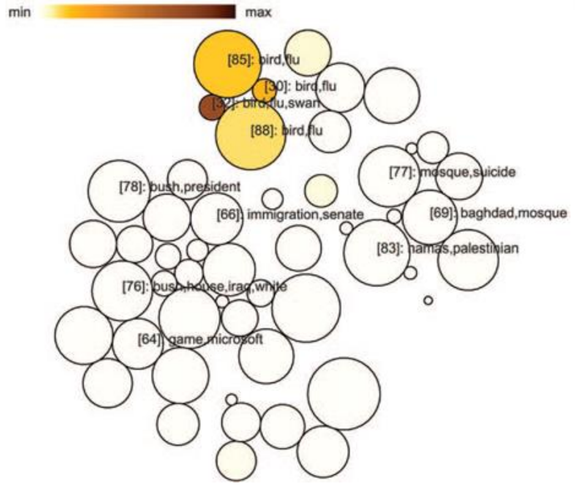
Typical scenarios

- Embeddings generated by deep neural models and LLMs
- Large text/image/audio audio datasets

Multidimensional Projection

Map data set onto the 2D plane, allowing direct exploration





Paulovich and Minghim, HiPP: a novel hierarchical point placement strategy and its application to the exploration of document collections, ***IEEE Trans. Visualization & Computer Graphics***, *2008*

What is dimensionality reduction?

A typical Machine Learning problem involves datasets described by thousands of features: very high embedding dimensionality

High dimensionality brings along many problems (*dimensionality curse*), e.g:

- Makes the training extremely slow
- Makes it difficult to find a good solution

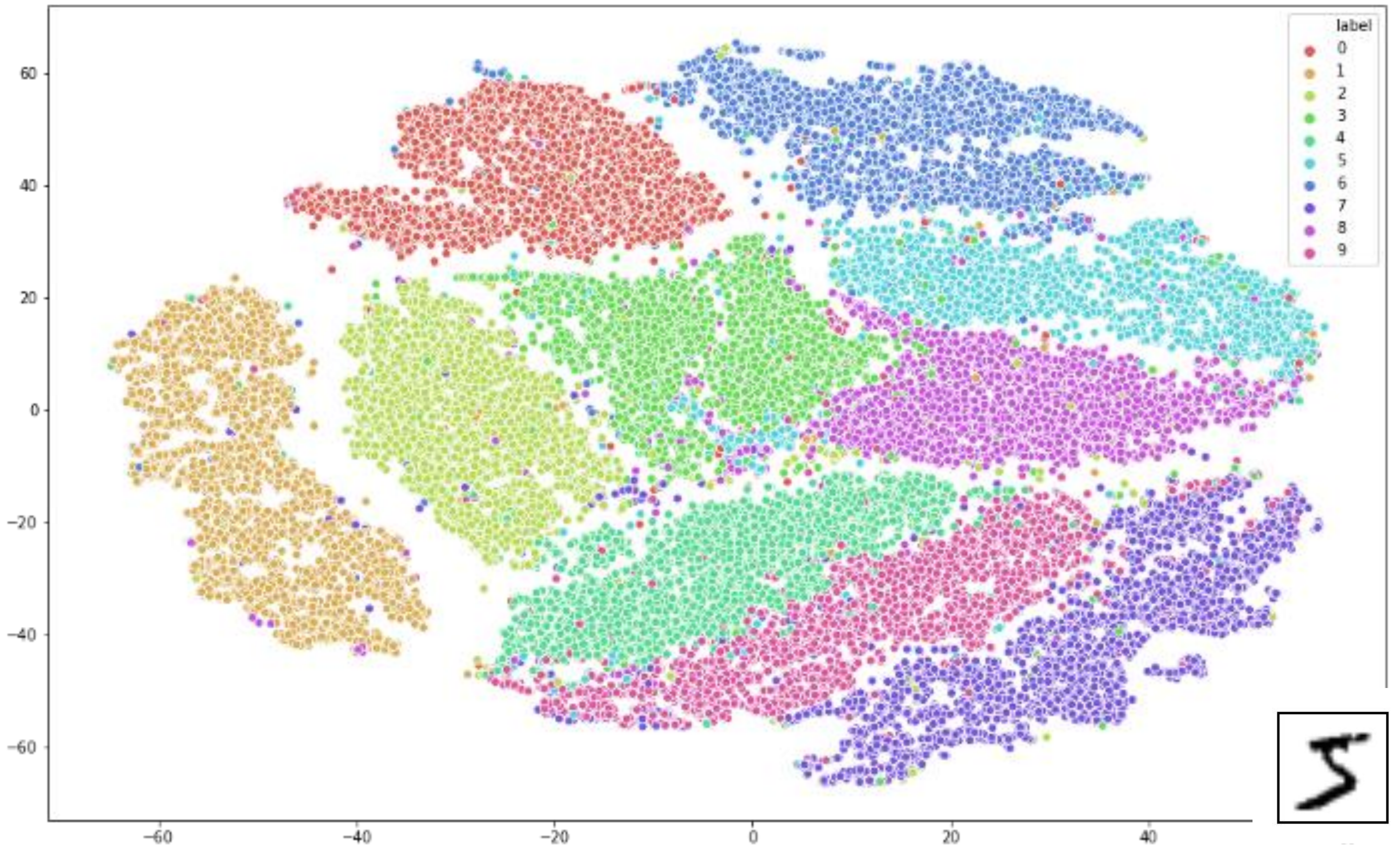
In simple terms, **Dimensionality Reduction** is the process of reducing the number of features to the most relevant ones.

What is dimensionality reduction?

Most Dimensionality Reduction algorithms are used for:

- Data Compression
- Noise Reduction
- Data Classification
- **Data Visualization**: drop the dimensionality down to two or three

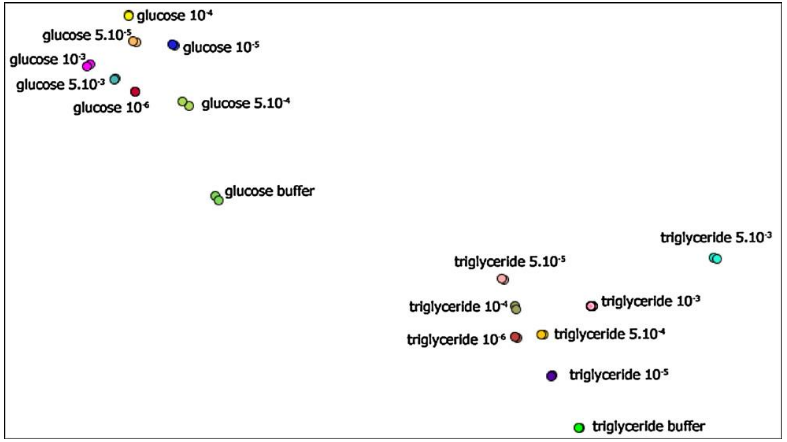
Make it possible to visualize the data on a 2d or 3d plot, meaning important insights can be gained by analysing the visible patterns, e.g. clusters, outliers, etc.

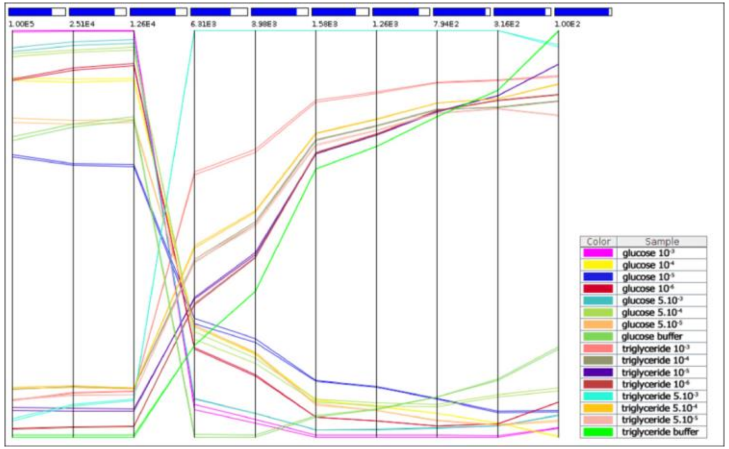


2D scatter plot of MNIST data after applying PCA (50 components) and then t-SNE.

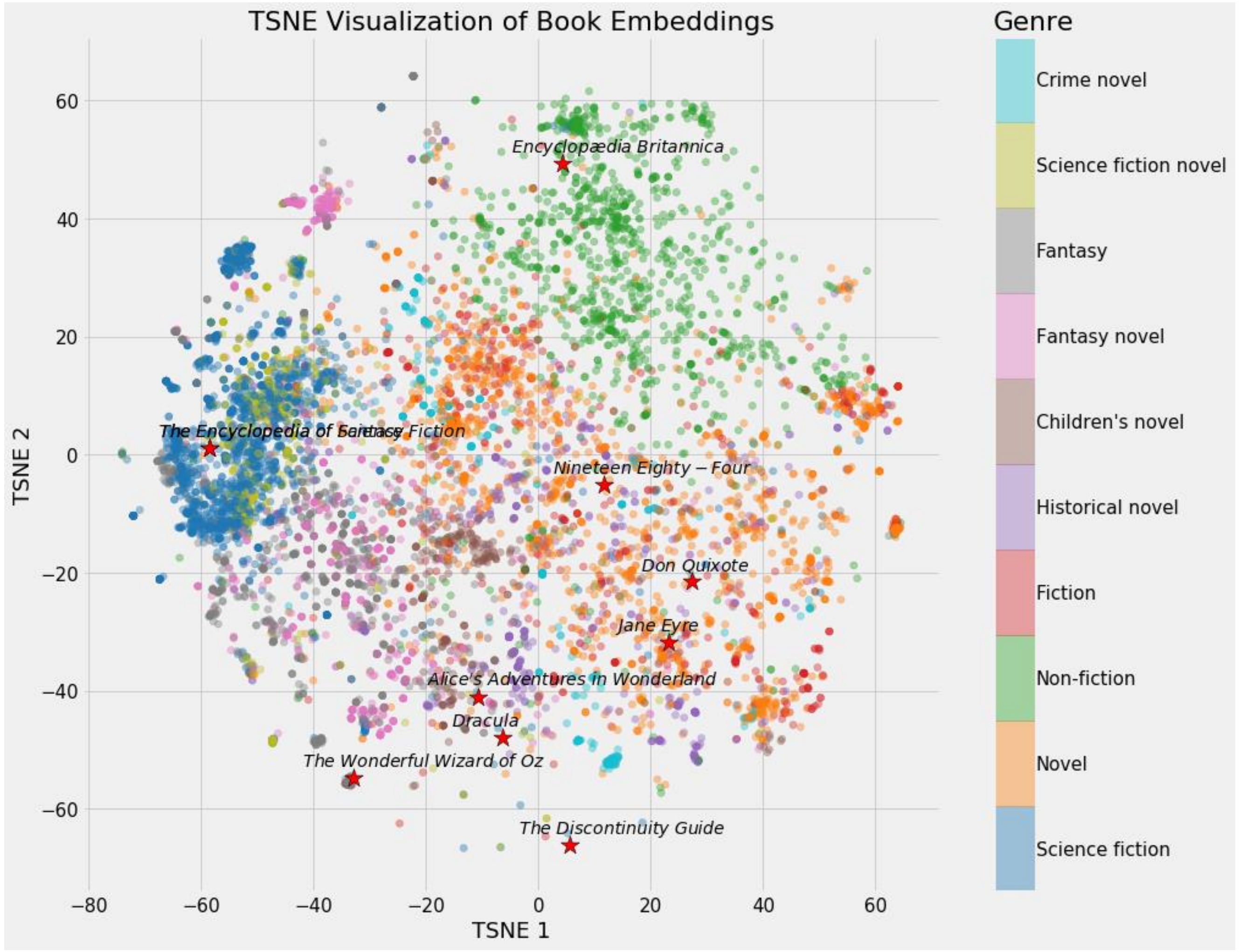
https://towardsdatascience.com/dimensionality-reduction-using-t-distributed-stochastic-neighbor-embedding-t-sne-on-the-mnist-9d36a3dd4521

Ex. Projection + Parallel coordinates





Source: Moraes et. al. Detection of glucose and triglycerides using information visualization methods to process impedance spectroscopy data, ***Sensors & Actuators B***, 2012



https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

Dimension reduction

- Feature selection
- Matrix factorization, e.g. PCA
- Distance error optimization, e.g., MDS, IDMAP
- Neighbor graphs, e.g. t-SNE, UMAP, LSP

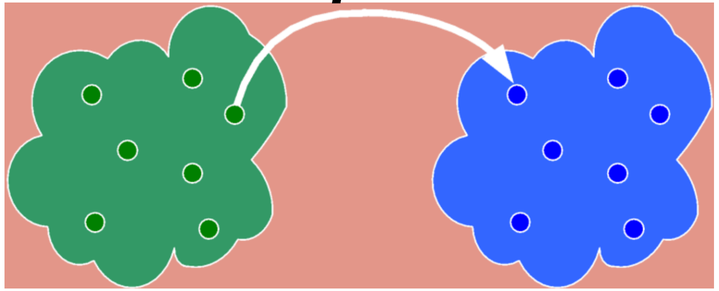
Multidimensional projections

... a way to `look into the data´ in scenarios of data described by too many features

when classical multidimensional visualization techniques are not effective as a starting point

Projection: concept

$\mathbf{X} \in R^m \quad f \quad \mathbf{Y} \in R^{p=\{2,3\}}$



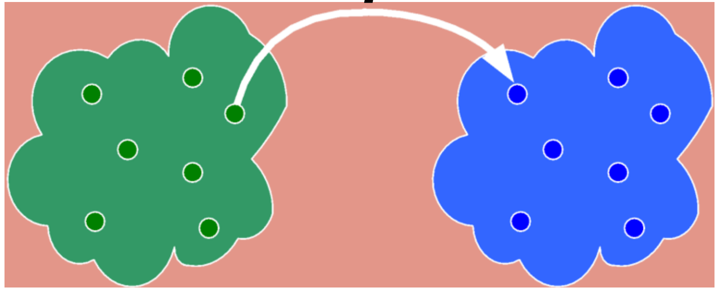
A mapping function *f* defines a spatial placement of the data in a visual space: 2 or 3 dimensions

How? Which properties a solution should satisfy?

Many many approaches are possible!

Multidimensional projection

$$\mathbf{X} \in R^m \quad f \quad \mathbf{Y} \in R^{p=\{2,3\}}$$

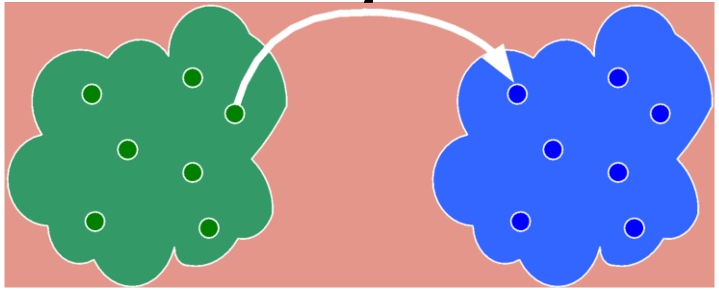


δ: $\mathbf{x}_i$, $\mathbf{x}_j \rightarrow R$, $\mathbf{x}_i,\mathbf{x}_j \in \mathbf{X}$ - a dissimilarity function (m-d)

d: $\mathbf{y}_i$, $\mathbf{y}_j \rightarrow R$, $\mathbf{y}_i,\mathbf{y}_j \in \mathbf{Y}$ - a distance function (2-d, 3-d) (Euclidean)

Multidimensional projection

$$\mathbf{X} \in R^m \quad f \quad \mathbf{Y} \in R^{p=\{2,3\}}$$

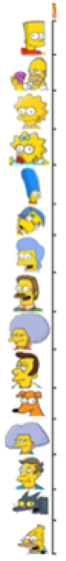


f: $\mathbf{X} \rightarrow \mathbf{Y}$, $\mathbf{y}_i = f(\mathbf{x}_i)$, $\mathbf{y}_j = f(\mathbf{x}_j)$

f attempts to minimize some error measure formulated as a difference between the pairwise point distances in the original m-d space and the 2-d projected space: $\Delta = |\delta(\mathbf{x}_i,\mathbf{x}_j) - d(\mathbf{y}_i, \mathbf{y}_j)| \approx 0, \forall\ \mathbf{x}_i,\mathbf{x}_j \in \mathbf{X}$

Data representation

Data set $\mathbf{X}_{nxm}$

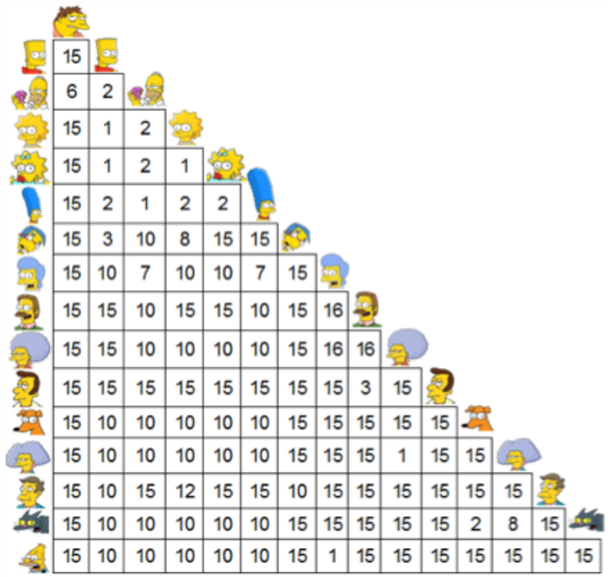


5	12	15	2	7	5	0	12	9	0	8
12	5	0	12	12	12	12	12	18	12	12
0	1	05	10	15	12	8	12	9	11	5
0	12	01	12	9	0	12	10	5	5	12
12	8	05	12	12	12	8	12	9	12	12
10	12	0	11	10	2	7	12	2	16	7
5	6	8	12	12	15	12	6	9	17	0
7	12	05	0	12	12	10	17	9	12	12
2	10	05	15	12	1	12	10	9	8	2
12	12	7	12	0	12	0	12	10	12	12
6	12	05	17	12	10	12	12	9	12	8
12	10	2	12	1	12	12	11	6	0	12
1	12	05	12	12	16	2	12	9	12	0
10	0	12	12	9	12	0	10	12	12	8
0	12	1	12	12	5	1	7	11	12	12
8	2	11	10	7	12	5	12	15	10	0

dimensional embedding
m-dimensional feature space
(embedding space)

each data instance as a vector
of numerical attributes
(dense feature vectors)

Data representation

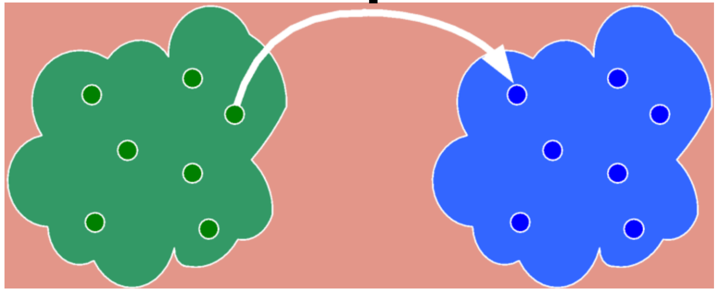


15														
6	2													
15	1	2												
15	1	2	1											
15	2	1	2	2										
15	3	10	8	15	15									
15	10	7	10	10	7	15								
15	15	10	15	15	10	15	16							
15	15	10	10	10	10	15	16	16						
15	15	15	15	15	15	15	15	3	15					
15	10	10	10	10	10	15	15	15	15	15				
15	10	10	10	10	10	15	15	15	1	15	15			
15	10	15	12	15	15	10	15	15	15	15	15	15		
15	10	10	10	10	10	15	15	15	15	15	2	8	15	
15	10	10	10	10	10	15	1	15	15	15	15	15	15	15

pairwise distances
between all data points

Projection strategy

$$\mathbf{X} \in R^m \quad f \quad \mathbf{Y} \in R^{p=\{1,2,3\}}$$



δ: $\mathbf{x}_i$, $\mathbf{x}_j \rightarrow R$, $\mathbf{x}_i$,$\mathbf{x}_j \in \mathbf{X}$

d: $\mathbf{y}_i$, $\mathbf{y}_j \rightarrow R$, $\mathbf{y}_i$,$\mathbf{y}_j \in \mathbf{Y}$

f: $\mathbf{X} \rightarrow \mathbf{Y}$, $|\delta(\mathbf{x}_i,\mathbf{x}_j) - d(f(\mathbf{x}_i), f(\mathbf{x}_j))| \approx 0$, $\forall$ $\mathbf{x}_i$,$\mathbf{x}_j \in \mathbf{X}$

Visualizing high-dimensional space

https://www.youtube.com/watch?v=wvsE8jm1GzE

see also

https://colah.github.io/posts/2014-10-Visualizing-MNIST/

Some techniques

PCA - Principal Component Analysis

MDS - Multidimensional Scaling

IDMAP - Interactive Document Map Minghim et al. VDA 2006

t-SNE - t-stochastic Neighbor Embedding

UMAP - Uniform Manifold Approximation and Projection

LSP - Least Squares Projection – Paulovich et al. 2008, IEEE TVCG

…

Principle Component Analysis PCA

well-known **unsupervised dimensionality reduction** technique

constructs **relevant** features/variables through linear (linear PCA) **combinations** of the original variables (features)

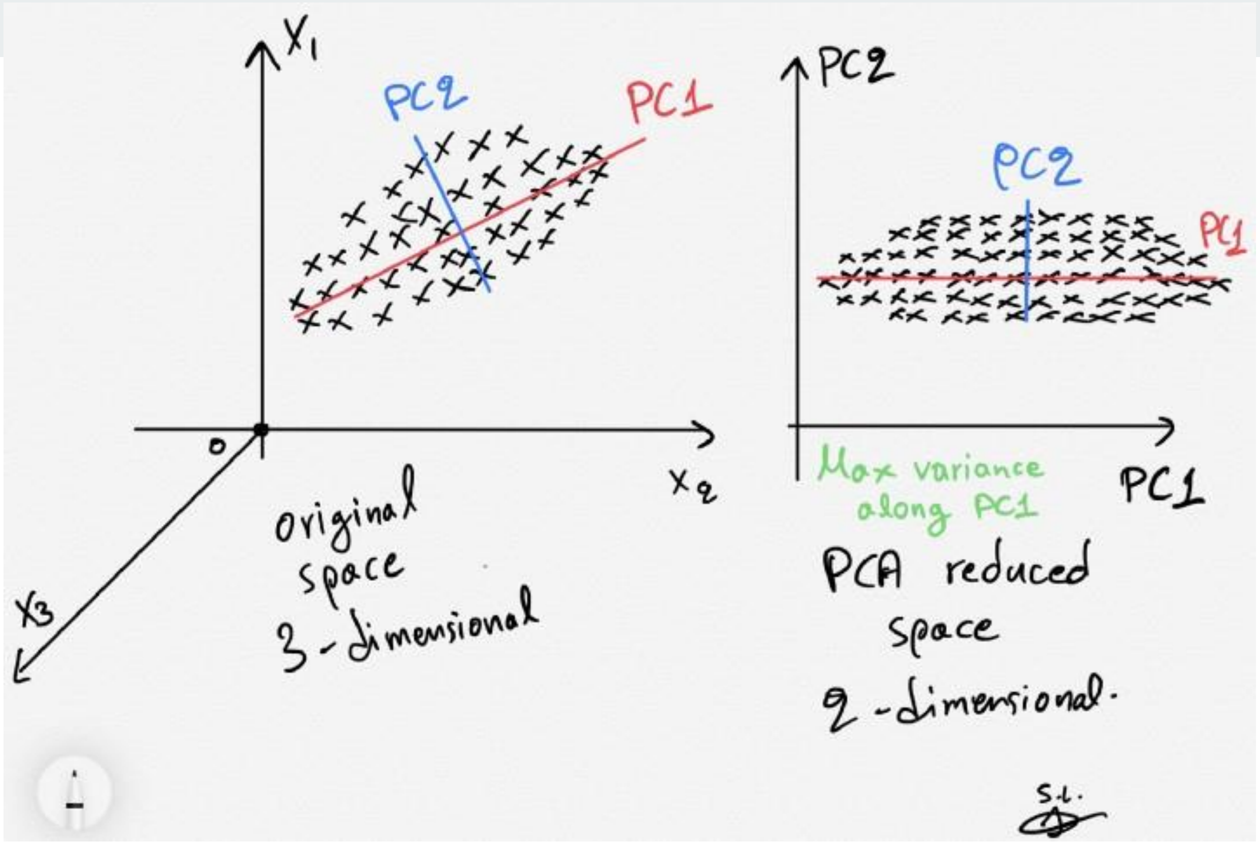
linearly transforms correlated variables into a smaller number of **uncorrelated** variables

projects the original data into the **reduced PCA space**

the resulting projected data are essentially **linear combinations** of the **original** data capturing most of the variance in the data

Principal Component Analysis (PCA)

https://medium.com/geekculture/pca-clearly-explained-when-why-how-to-use-it-and-feature-importance-a-guide-in-python-37596289571c


X_1
PC2
PC1
0
X_2
X_3
original space 3-dimensional
PC2
PC2
PC1
PC1
Max variance along PC1
PCA reduced space 2-dimensional.

PCA is an **orthogonal transformation** of the original data into a reduced PCA space.... such that the first component explains the most variance in the data with each subsequent component explaining less

When/Why use PCA

useful in processing data where **multi-colinearity** exists between the **features/variables**

when **the dimensions of the input features are high** (e.g. a lot of variables)

can be also used for **denoising** and **data compression**

PCA Steps

1. the original input variables stored in **X** are **z-scored** such each original variable (column of **X**) has zero mean and unit standard deviation

2. construct the **covariance** matrix $\mathbf{C(X)= (1/n)}*\mathbf{X}^{\mathbf{T}}*\mathbf{X}$ (in case of z-scored data the covariance is equal to the correlation matrix)

PCA Steps

3. Get the eigenvalue/eigenvector decomposition of **C(X)**.

Sort eigenvalues in **decreasing** order, representing decreasing variance in the data

- the eigenvalues are equal to the variance, the corresponding eigenvectors give the directions of variance

PCA Steps

4. Multiply **the originally normalized data** by the PCs (the **leading eigenvectors** of the covariance matrix), to **project** the **original (normalized) data** points onto the **reduced PCA space**

$$Y = T^T * X$$

Example: 2-dimensional data

Eigenvectors and corresponding eigenvalues

$$v1 = \begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix} \qquad \lambda_1 = 1.284028$$

$$v2 = \begin{bmatrix} -0.7351785 \\ 0.6778736 \end{bmatrix} \qquad \lambda_2 = 0.04908323$$

PC1 = v1 explains 96% of variance **PC2 = v2** explains ~4% variance

Example: 2-dimensional data

Transformation matrix T

$$\begin{bmatrix} 0.6778736 & -0.7351785 \\ 0.7351785 & 0.6778736 \end{bmatrix}$$

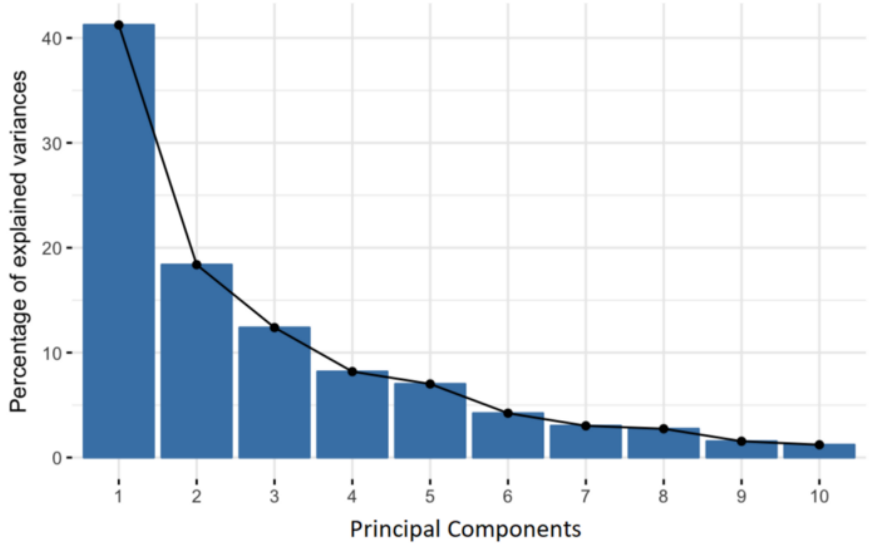
2D

$$\begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix}$$

1D

Scree plot

Data set with 10 dimensions:
10 PCs, ranked by importance
(% explained variance)



Source: https://builtin.com/data-science/step-step-explanation-principal-component-analysis

PCA Feature importance

The **importance** of each **feature/variable** for a PC is reflected by the **magnitude** of the **corresponding values in the eigenvectors** (higher magnitude — higher importance).

Loading factors: the coefficients of the linear combination of the original variables from which the principal components (PCs) are constructed.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn import decomposition
from sklearn import datasets
from sklearn.preprocessing import scale

# load iris dataset
iris = datasets.load_iris()

x = scale(iris.data)
y = iris.target

# apply PCA
pca = decomposition.PCA(n_components=2)
X = pca.fit_transform(X)
```

```
loadings = pd.DataFrame(pca.components_.T,
columns=['PC1', 'PC2'], index=iris.feature_names)
loadings

                        PC1         PC2
sepal length (cm)   0.521066    0.377418
sepal width (cm)   -0.269347    0.923296
petal length (cm)   0.580413    0.024492
petal width (cm)    0.564857    0.066942
```

PCA explained@StaQuest

https://www.youtube.com/watch?v=HMOI_lkzW08   ~5 min – how to interpret

https://www.youtube.com/watch?v=FgakZw6K1QQ ~21 min – step by step

https://www.youtube.com/watch?v=oRvgq966yZg   ~8 min – practical tips

PCA explained

Gewers et al. Principal Component Analysis: A Natural Approach to Data Exploration. *ACM Computing Surveys* 54(4)- https://dl.acm.org/doi/10.1145/3447755

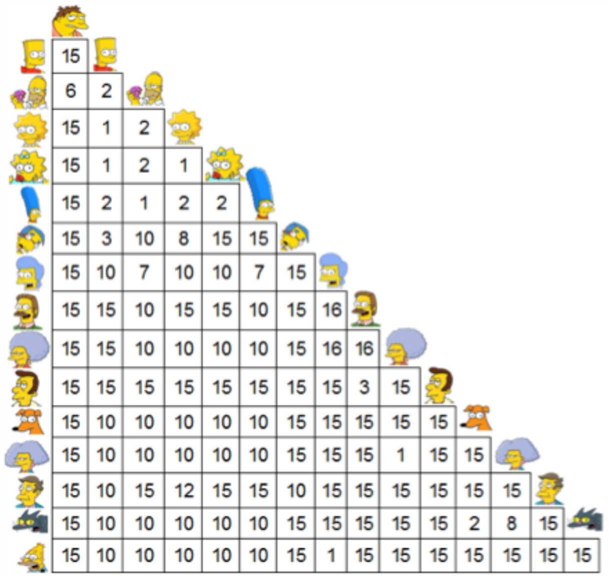
Multidimensional Scaling (MDS)

The input to MDS are the pairwise distances between the high-dimensional data instances: the goal is to preserve these distances in a lower dimensional space without much loss of information (Cox and Cox 2000).

The “classical” (metric) multidimensional scaling algorithm seeks to minimize a stress function:

$$Stress_{\delta} = \left(\frac{\sum_{i,j} \left(\delta_{i,j} - \| y_i - y_j \| \right)^2}{\sum_{i,j} {\delta_{i,j}}^2} \right)^{1/2}$$

Data representation

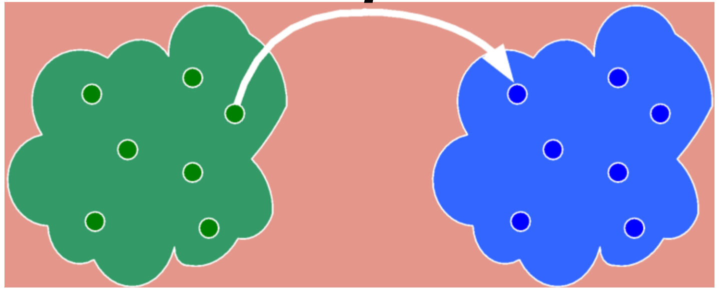


15														
6	2													
15	1	2												
15	1	2	1											
15	2	1	2	2										
15	3	10	8	15	15									
15	10	7	10	10	7	15								
15	15	10	15	15	10	15	16							
15	15	10	10	10	10	15	16	16						
15	15	15	15	15	15	15	15	3	15					
15	10	10	10	10	10	15	15	15	15	15				
15	10	10	10	10	10	15	15	15	1	15	15			
15	10	15	12	15	15	10	15	15	15	15	15	15		
15	10	10	10	10	10	15	15	15	15	15	2	8	15	
15	10	10	10	10	10	15	1	15	15	15	15	15	15	15

pairwise distances
between all data points

Multidimensional projection

$\mathbf{X} \in R^m \quad f \quad \mathbf{Y} \in R^{p=\{2,3\}}$



f: $\mathbf{X} \rightarrow \mathbf{Y}$, $\mathbf{y}_i = f(\mathbf{x}_i)$, $\mathbf{y}_j = f(\mathbf{x}_j)$

f attempts to minimize some error measure formulated as a difference between the pairwise point distances in the original m-d space and the 2-d projected space: $\Delta = |\delta(\mathbf{x}_i,\mathbf{x}_j) - d(\mathbf{y}_i, \mathbf{y}_j)| \approx 0, \forall\, \mathbf{x}_i,\mathbf{x}_j \in \mathbf{X}$

Multidimensional Scaling (MDS)

Result is equivalent to PCA if the input distances are Euclidean distances!

MDS is hard to scale to large data sets (why?)

It rarely produces meaningful/useful embeddings of large high-dimensional data (why?)

“Modern” techniques adopt strategies that attempt to preserve neighborhoods, rather than global distances

T-SNE, UMAP

- Compute high-dimensional probabilities *p* and low-dimensional probabilities *q*
    - *p* and *q* are probability distributions of neighborhoods
    - $p_{i|j}$ probability that a data point *j* would pick data point *i* as neighbor
    - If points y_i are placed correctly in low-dimensional space, the conditional probabilities *p* and *q* will be very similar

- Use a cost function to calculate the difference between probabilities

t-SNE

t-distributed Stochastic Neighbor Embedding. Introduced by Laurens van der Maaten & Geoffrey Hinton 2008: http://www.cs.toronto.edu/~hinton/absps/tsne.pdf

a. **Stochastic** → not definite but random probability (generates slightly different results each time on the same data set)
b. **Neighbor** → concerned only about retaining the variance of neighbor points (preserve original neighborhoods, or clusters)
c. **Embedding** → finds a representation of the data into lower dimension

t-SNE

attempts to find a projection of the data that preserves the data clusters that exist in the high-dimensional space (i.e., only the small distances are preserved!)

a very good explanation at StatQuest:

https://www.youtube.com/watch?v=NEaUSP4YerM&ab_channel=StatQuestwithJoshStarmer

t-SNE

Step 1: computes probability distribution of the pairwise distances in high dimension such that similar (closer) objects are assigned a higher probability and dissimilar objects are assigned lower probability.

Step 2: replicates the same probability distribution on lower dimensions iteratively till the Kullback-Leibler divergence is minimized.

Kullback-Leibler (KL) *divergence*: a measure of the difference between the probability distributions from Step 1 and Step 2.

t-SNE

p_{ij}: affinity (similarity) between data samples *i* and *j* in high-dimensional space

q_{ij}: affinity (similarity) between data samples *i* and *j* in low-dimensional space

similarities *p* and *q* are defined such that they sum to 1

$$KL = \sum_{i,j} p_{ij} log \frac{p_{ij}}{q_{ij}}$$

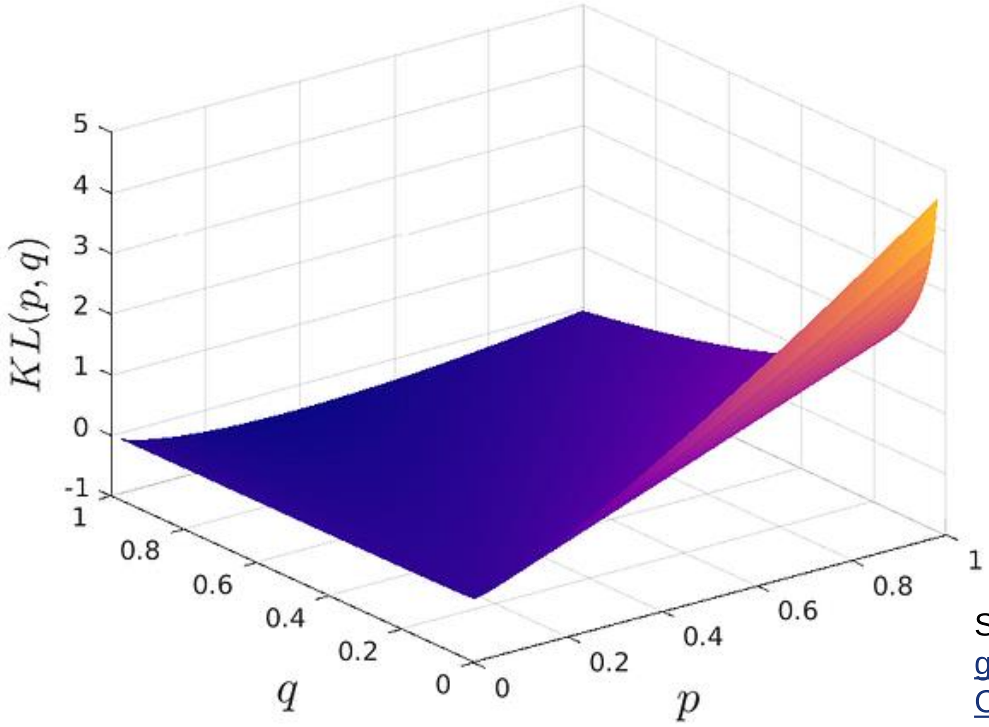
starts with random configuration of the points in low-dimensional space, optimizes their positions with gradient descent in order to minimize KL

t-SNE

Kullback-Leibler divergence as a measure of the mismatch between the high-dimensional and the low-dimensional condition probabilities, for each data point

The algorithm starts by placing all the y_i in random locations, and then is trained to minimize the cost function using gradient descent, i.e., optimizes their positions with gradient descent in order to minimize the KL divergence

Kullback-Leibler divergence



Close points (similar instances, high *p):* strongly penalizes mapping them far apart (low *q*)

good preservation of close neighborhoods

global structure is not well preserved

Source[Visualizing Your Embeddings. An evolutionary guide from SNE to t-SNE… | by Francisco Castillo Carrasco | Towards Data Science]

t-SNE

two main parameters

1. *n_iter*: the number of iterations
2. *perplexity*: can be interpreted as a guess on the number of close neighbors each point has (default is 30) (neighborhood density)

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

t-SNE

t-SNE plots may vary a lot depending on the choice of parameters: recommendation is to run with different settings and analyze which works best

as the technique is stochastic, each run may produces slightly different results. *sklearn* allows avoiding this by fixing parameter *random_state*

the initial configuration can have a large effect on the result (the loss function may have many local minima) (use informative initialization)

t-SNE's priority is to preserve neighborhoods (small distances, or local structures): it often does not preserve global structures, e.g., the relative positions of the clusters, or the cluster's distances

UMAP

Uniform Manifold Approximation and Projection

(at the core) very similar to t-SNE in how it works, but it is more efficient (increased speed) and preserves the data's global structure better (better balance between local/global)

Scales well relative to data set size and dimensionality

Strong theoretical foundation (manifold theory and topological data analysis)

Key step: construct a neighborhood graph from the data in the high-dimensional space

UMAP

very good explanation here: https://pair-code.github.io/understanding-umap/

and here (StatQuest): https://www.youtube.com/watch?v=eN0wFzBA4Sc

UMAP

Parameters

number of neighbors: the number of approximate nearest neighbors used to construct the initial high-dimensional graph. Controls how UMAP balances local versus global structure - low values will push UMAP to focus more on local structure, while high values will push UMAP towards representing the big-picture structure while losing fine detail.

minimum distance: the minimum distance between points in low-dimensional space. This parameter controls how tightly UMAP clumps points together, with low values leading to more tightly packed embeddings, whereas larger values will make UMAP pack points together more loosely, focusing instead on the preservation of the broad topological structure.

UMAP

Uses cross-entropy as loss function

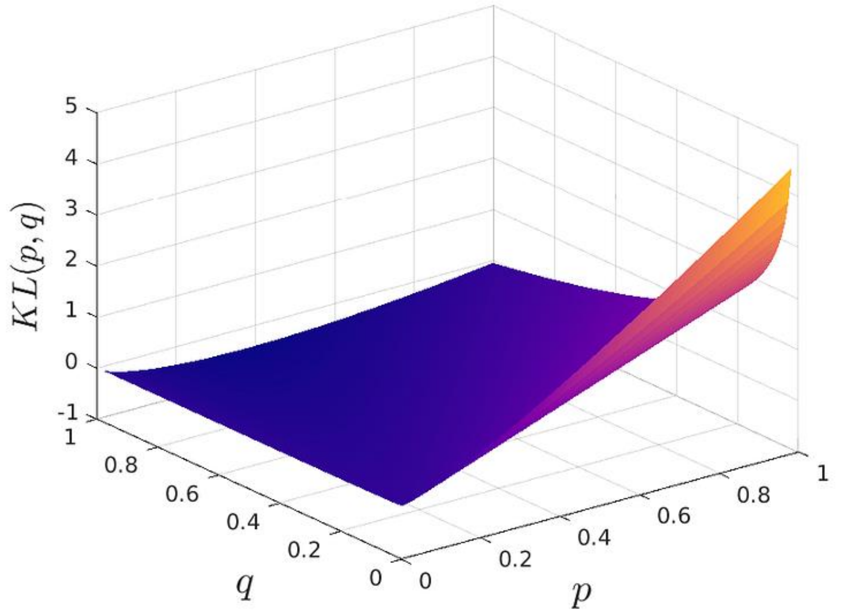
CE loss function has both attractive and repulsive forces

UMAP uses stochastic gradient descent to minimize the cost function, instead of the slower gradient descent

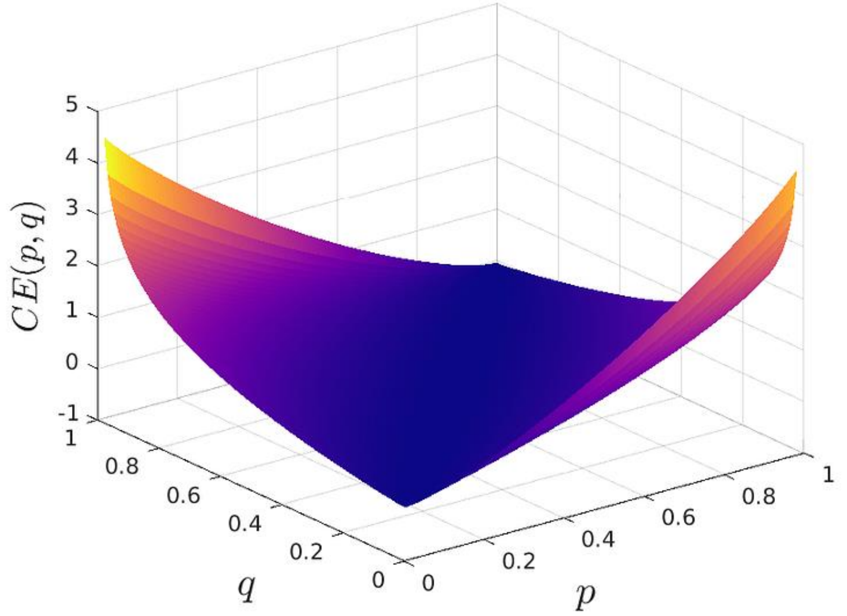
“With this new choice of loss function, placing objects that are far away in high-dimensional space nearby in the low-dimensional space is penalized. Thanks to the better choice of loss function, UMAP can capture more of the global structure than its predecessors.”

Visualizing Your Embeddings. An evolutionary guide from SNE to t-SNE… | by Francisco Castillo Carrasco | Towards Data Science

Cross entropy loss function



KL loss used by t-SNE



CE loss used by UMAP

Source: Visualizing Your Embeddings. An evolutionary guide from SNE to t-SNE… | by Francisco Castillo Carrasco | Towards Data Science

UMAP

Uses spectral initialization, not random initialization of the low-dimensional points (Laplacian Eigenmaps initialization)

Quick to compute, good starting point for stochastic gradient descent, in theory deterministic (but computational results approximate, thus determinism not guaranteed)

But good stability: initialization provides faster convergence as well as greater consistency, i.e., multiple runs of UMAP will yield similar results.

Visualizing Your Embeddings. An evolutionary guide from SNE to t-SNE… | by Francisco Castillo Carrasco | Towards Data Science

PCA/UMAP/t-SNE in action

See

> https://projector.tensorflow.org/

Discussion…

projections: preserve *distances* vs *neighborhoods*

$$|\delta(\mathbf{x}_i,\mathbf{x}_j) - d(f(\mathbf{x}_i), f(\mathbf{x}_j))| \approx 0, \forall\, \mathbf{x}_i,\mathbf{x}_j \in \mathbf{X}$$

preserving **all** distances **equally** is not feasible (why?), and the neighborhood of points is a relative concept

Discussion: projections vs PCA

PCA is simple and easy to use, quick to compute

captures the global variance in the data, linear correlation

does not capture local variance, non linear correlations

scatterplot 2D: axes *x* and *y* associated with the first and second highest PCs, possible to relate the original attributes with the PCs.

Projections in general do not assume linearly correlated attributes

"*scatterplot*" 2D (t-SNE, IDMAP) is not actually a *scatterplot*, but a "similarity map" of the data instances: position of points along *x* and *y* axes do not have an interpretable meaning

Discussion: projections vs PCA

PCA is a *parametric* mapping: a function is computed that is applied to project the data

Most projections techniques are *non parametric*, in that no explicit mapping function is computed

Discussion: projections vs PCA

PCA and the projections we discussed are *non-supervised*, in the sense that they do not use the class structure to compute the low dimensional embeddings

Additional material

PCA

https://towardsdatascience.com/pca-clearly-explained-how-when-why-to-use-it-and-feature-importance-a-guide-in-python-7c274582c37e

Additional material

t-SNE

https://www.youtube.com/watch?v=NEaUSP4YerM&ab_channel=StatQuestwithJoshStarmer

https://towardsdatascience.com/what-why-and-how-of-t-sne-1f78d13e224d

https://colah.github.io/posts/2014-10-Visualizing-MNIST/

Visualizing Your Embeddings. An evolutionary guide from SNE to t-SNE… | by Francisco Castillo Carrasco | Towards Data Science

Additional material

UMAP @ StatQuest

- https://www.youtube.com/watch?v=eN0wFzBA4Sc&t=942s (main ideas)
- https://www.youtube.com/watch?v=jth4kEvJ3P8 (mathematical details)

https://pair-code.github.io/understanding-umap/

How to assess/compare?

compute stress (error function) – cannot compare across techniques , as different techniques optimize different functions

The goodness of fit criteria are measured on different scales and use different methodologies

Ideally , should have a method independent measure of solution fit

How to assess/compare?

verify preservation of neighborhoods relative to original space

verify group/class segregation (on data with known groups/classes)

compare distributions of pairwise distances in the original space vs in the projected space

How to assess/compare?

verify preservation of neighborhoods

given a data point $\mathbf{x}_i$ in the original space, who are the k-nearest neighbors of $\mathbf{x}_i$? (k = 1, k = 2, k = 3, ...)

given the corresponding data point in the projected space $\mathbf{y}_i$, who are the are the k-nearest neighbors of $\mathbf{y}_i$?

How to assess/compare?

Neighborhood Hit - F. V. Paulovich Mapeamento de dados multi-dimensionais - integrando mineração e visualização. Tese de doutorado ICMC-USP, 2008

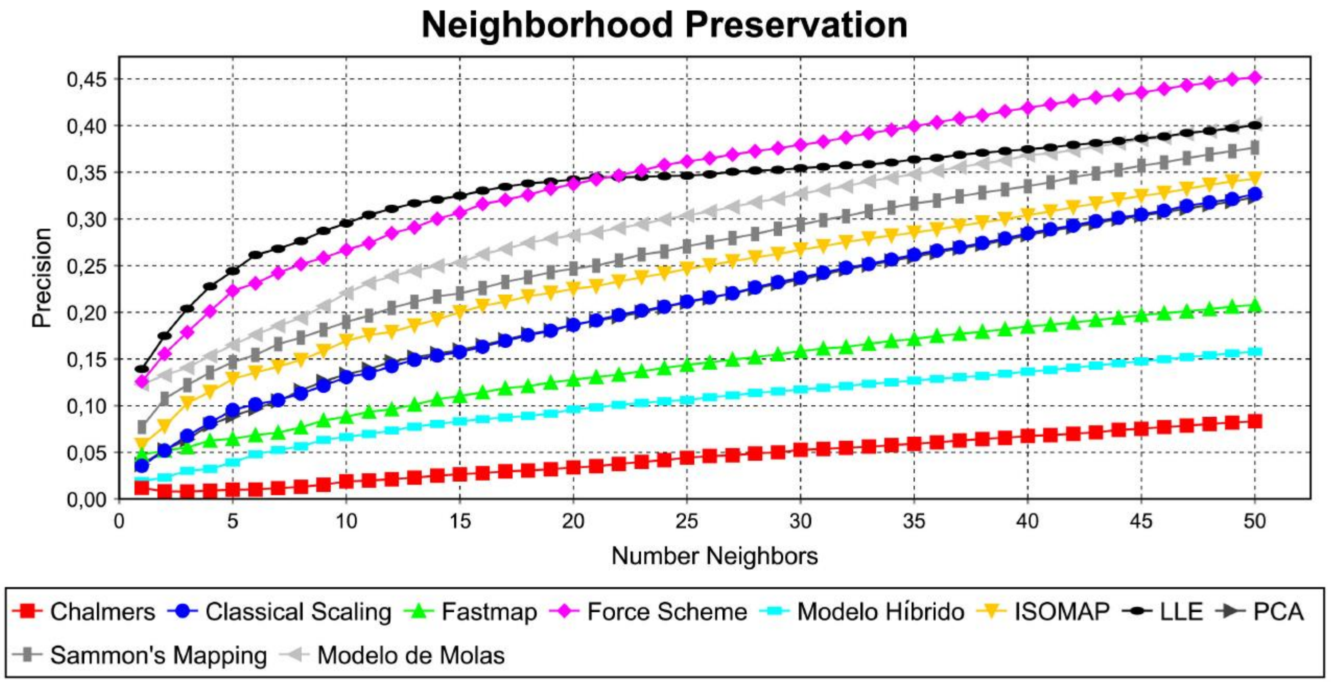


Figura 2.19: Avaliação das projeções criadas nas seções anteriores usando-se a técnica *Neighborhood Preservation*. Usando-se essa análise e a anterior é possível definir a técnica que melhor consegue separar as classes existentes nos dados e preservar a vizinhança dos objetos nos pontos projetados.

Fonte: F. V. Paulovich Mapeamento de dados multi-dimensionais - integrando mineração e visualização. Tese de doutorado ICMC-USP, 2008

Nieghborhood preservation

Curves give information on average preservation for varying neighborhood sizes, but not in a point by point basis

Alternatives?

show in the projection itself v. Martins et al. 2015. Explaining Neighborhood Preservation for Multidimensional Projections

show as a heatmap of item vs neighborhood preservation (requires ordering the items according to some criteria...)

How to assess/compare?

Groups/classes known: Silhouette Coefficient (originally used to validate results of clustering algorithms).

https://en.wikipedia.org/wiki/Silhouette_(clustering) SC in [-1,+1]

Compare SC in original and (e.g., different) projected spaces

Additional material

F. V. Paulovich Mapeamento de dados multi-dimensionais - integrando mineração e visualização. Tese de doutorado ICMC-USP, 2008