Unsupervised Learning – Part 3

ESM3081 Programming for Data Science

Seokho Kang

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

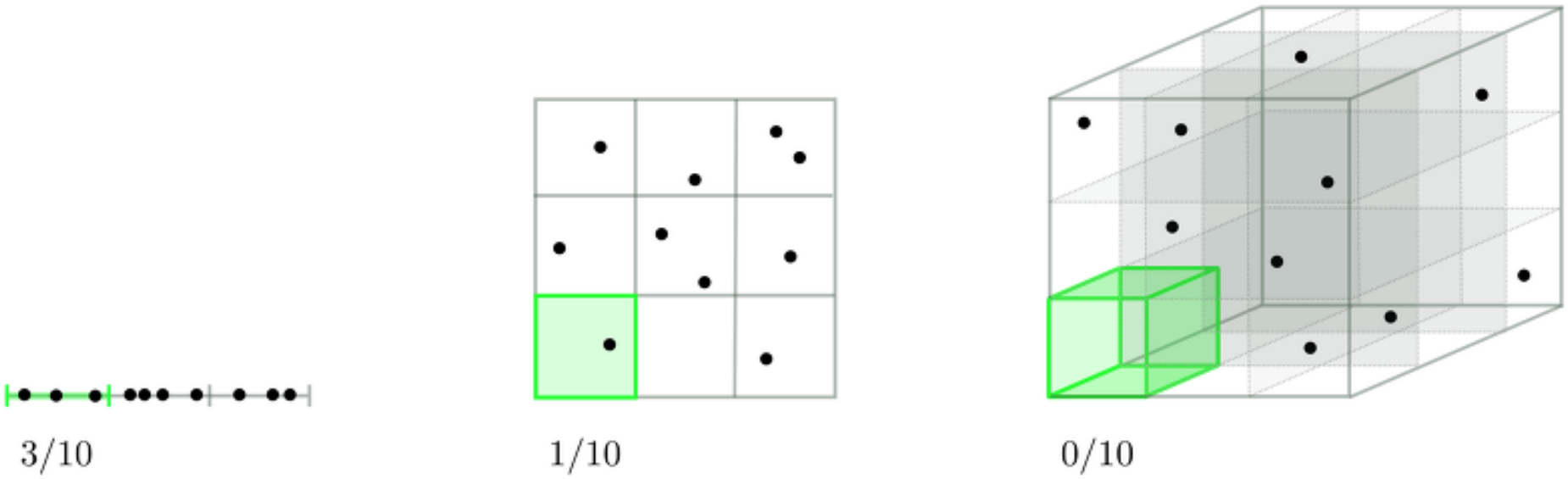
Learning algorithms covered in this course

- **Unsupervised Learning**
  - **Dimensionality Reduction & Visualization**
    - (Projection) Principal Component Analysis (PCA)
    - **(Manifold Learning) t-distributed Stochastic Neighbor Embedding (t-SNE)**
    - …
  - **Clustering**
    - K-Means
    - Hierarchical Clustering
    - DBSCAN
    - …

t-distributed Stochastic Neighbor Embedding

The Curse of Dimensionality

- **High-dimensional datasets are at risk of being very sparse**: most training data points are likely to be far away from each other.
- **One solution to the curse of dimensionality could be to increase the size of the training set to reach a sufficient density of training data points**



Dimensionality Reduction for Machine Learning

- Many machine learning problems involve thousands or even millions of features for each training data point.
  - It makes training extremely slow.
  - It makes it much harder to find a good solution.
  - Much more training data is needed.

- In real-world problems, it is often possible to reduce the number of features considerably, turning an intractable problem into a tractable one without losing much information.
  - **Example**: MNIST images
    - The pixels on the image borders are almost always white, so you could completely drop these pixels from the training set. (superfluity)
    - Two neighboring pixels are often highly correlated, so you could merge them into a single pixel. (redundancy)

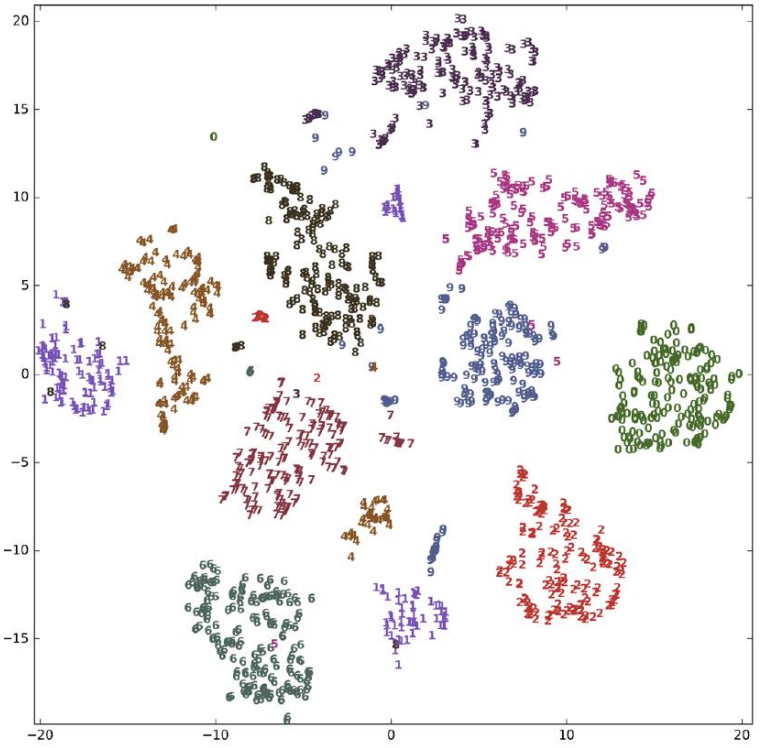
Dimensionality Reduction for Visualization

- Dimensionality reduction is extremely useful for **data visualization**.
  - Reduce the number of dimensions down to two (or three) so that we can plot a high-dimensional training set on a graph (scatterplot).
  - Gain some important insights by visually detecting patterns, such as clusters.

High-dimensional data

Dimensionality reduction

Low-dimensional visualization



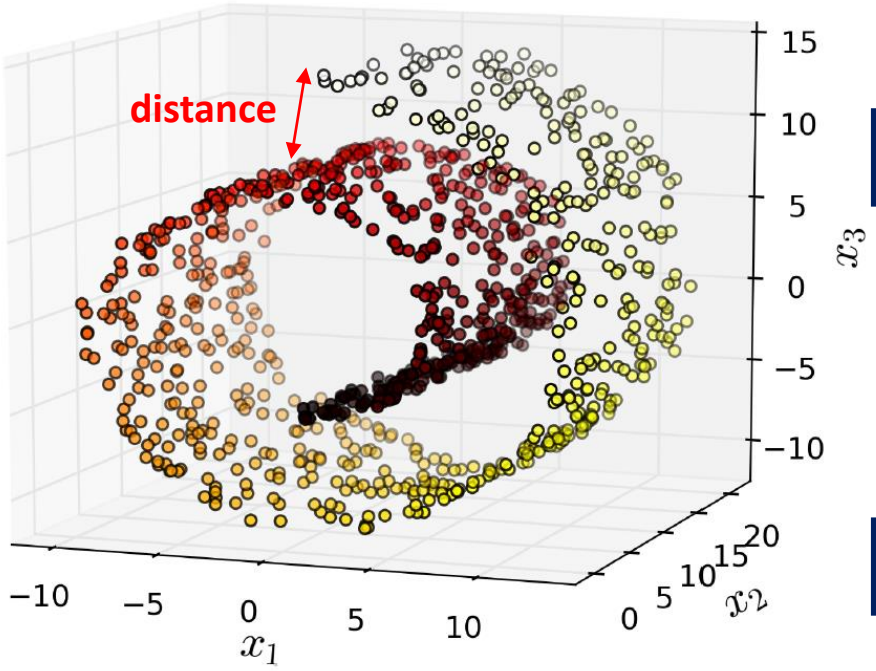
Dimensionality Reduction Approaches

- **Projection (Linear Dimensionality Reduction)** (e.g., PCA, …)
  - **Assumption**: The data lie within (or close to) a much lower-dimensional *subspace* of the high-dimensional space. Training data points are *not* spread out uniformly across all dimensions: Many features are almost constant, while others are highly correlated.
  - Projection algorithms choose an "interesting" linear projection of the data. These methods can be powerful, but often miss important non-linear structures in the data.

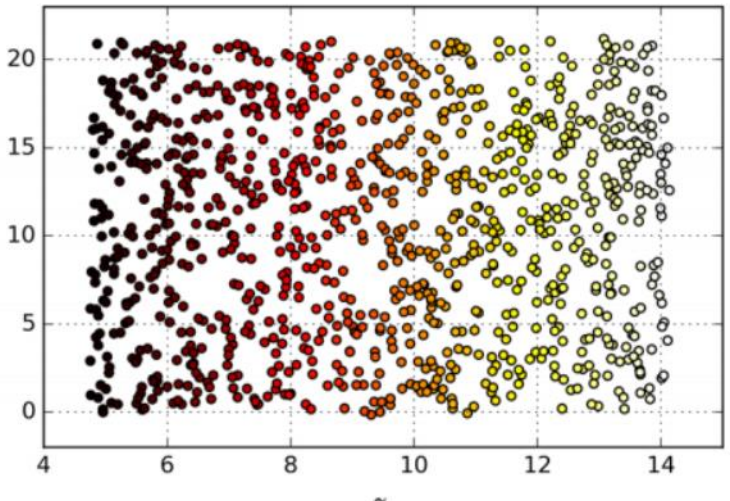
- **Manifold Learning (Nonlinear Dimensionality Reduction)** (e.g., t-SNE, …)
  - **Assumption**: The data lie within (or close to) a much lower-dimensional *manifold*. This assumption is very often empirically observed.
  - Manifold learning algorithms work by modeling the *manifold* on which the training data points lie.
  - They allow for much more complex mappings, and often provide better visualizations, especially for non-linear structures of the data.

Dimensionality Reduction Approaches

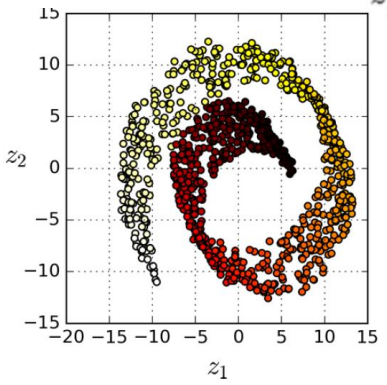
- **Example: *Swiss roll* dataset**
  - The *Swiss roll* dataset is an example of a 2D *manifold* (a 2D shape that is bent and twisted in a 3D space).
  - Projection algorithms perform worse in many cases the subspace may twist and turn.



unrolling



PCA



Euclidean vs geodesic distances?

Dimensionality Reduction Approaches

- **Notes on Manifold Learning**
  - Manifold learning algorithms are mainly aimed at visualization, and so are rarely used to generate more than two or three new features.
  - Manifold learning can be useful for exploratory data analysis, but is rarely used if the final goal is supervised learning.
  - The typical manifold learning problems are unsupervised: they learn the high-dimensional structure of the data from the data itself without the use of *class labels.*

t-Distributed Stochastic Neighbor Embedding

- *t-Distributed Stochastic Neighbor Embedding* (t-SNE) reduces dimensionality while trying to keep similar data points close and dissimilar data points apart, based solely on how close points are in the original space.
- The idea behind t-SNE is to find a low-dimensional representation of the data that preserves the distances between points as best as possible.
  - t-SNE starts with a random representation for each data point
  - Then, it tries to make …
    - points that are close in the original feature space closer (more emphasis on this)
    - points that are far apart in the original feature space farther apart
- t-SNE is mostly used for ***visualization***, in particular to visualize clusters of data points in high-dimensional space. It doesn't allow transformations of new data.

t-Distributed Stochastic Neighbor Embedding

- **Pseudocode**

  Given a (training) dataset $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of *d* features

  Goal: Find r-dim representation that best preserves the distances between points ($r \ll d$)

  1. Compute pairwise similarities p_{ij} for original data points in the training dataset D
  2. Initialize r-dim representation $\boldsymbol{Z}^{(0)} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_n\}$ by sampling from $N(\mathbf{0}, \epsilon \boldsymbol{I})$, $\epsilon > 0$
  3. Update the representation for T iterations. At the t-th iteration:
     - Compute pairwise similarities q_{ij} for $\boldsymbol{Z}^{(t-1)}$
     - Compute gradient $\nabla_{\boldsymbol{Z}} \mathrm{KL}(P||Q)$
     - Set $\boldsymbol{Z}^{(t)} = \boldsymbol{Z}^{(t-1)} + \eta \nabla_{\boldsymbol{Z}} \mathrm{KL}(P||Q) + \alpha(t)(\boldsymbol{Z}^{(t-1)} - \boldsymbol{Z}^{(t-2)})$

  * T: number of iterations (n_iter), η: learning rate, $\alpha(t)$: momentum

  Note: *It is highly recommended to use another dimensionality reduction method (e.g. PCA) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high.*

p.d.f. of $N(\mu, \sigma^2)$

$$f(x) = \frac{1}{\sqrt{2\pi\,\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

t-Distributed Stochastic Neighbor Embedding

- **Mathematical Details**
  - The similarity of the data point $\mathbf{x}_j$ to the data point $\mathbf{x}_i$ in the original space is measured by the conditional probability $p_{j|i}$ that $\mathbf{x}_i$ would pick $\mathbf{x}_j$ as its neighbor, if neighbors were picked in proportion to their probability density under a Normal distribution centered at $\mathbf{x}_i$.
    (*Note*: $p_{j|i}$ is the probability function of the conditional probability distribution P_i, $\sum_j p_{j|i} = 1$)

$$p_{j|i} = \begin{cases} \dfrac{\exp\left(-0.5\left\|\mathbf{x}_j - \mathbf{x}_i\right\|^2/\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-0.5\|\mathbf{x}_k - \mathbf{x}_i\|^2/\sigma_i^2\right)} & , \quad i \neq j \\ 0 & , \quad i = j \end{cases}$$

  - The variance σ_i^2 is determined such that the ***perplexity*** of the distribution P_i has a fixed value specified by the user. The perplexity can be interpreted as a smooth measure of the effective number of neighbors. It increases monotonically with σ_i^2.

$$\mathrm{Perp}(P_i) = 2^{H(P_i)} = 2^{-\sum_j p_{j|i}\log_2 p_{j|i}}$$

  - We define the pairwise similarity p_{ij} as below.
    (*Note*: p_{ij} is the probability function of the joint probability distribution P, $\sum_{i,j} p_{ij} = 1$)

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

van der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. Journal of Machine Learning Research, 9, 2579-2605.

p.d.f. of t(1)

$$f(x) = \frac{1}{\pi(1+x^2)}$$

t-Distributed Stochastic Neighbor Embedding

- **Mathematical Details**
  - Given $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$, t-SNE aims to r-dim representation $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_n\}$ (with $\boldsymbol{z}_i \in \mathbb{R}^r$, $r \ll d$) that reflects the similarities p_{ij} in the original space as well as possible.
  - Using a t-distribution with one degree of freedom (Cauchy distribution) as the heavy-tailed distribution, we measure the similarity between two points $\mathbf{z}_i$ and $\mathbf{z}_j$ in the low-dimensional space below.
    (*Note*: q_{ij} is the probability function of the joint probability distribution Q, $\sum_{i,j} q_{ij} = 1$)

$$q_{ij} = \begin{cases} \dfrac{\left(1 + \left\|\mathbf{z}_j - \mathbf{z}_i\right\|^2\right)^{-1}}{\sum_{k \neq l}(1 + \|\mathbf{z}_k - \mathbf{z}_l\|^2)^{-1}} & , \quad i \neq j \\ 0 & , \quad i = j \end{cases}$$

  - The minimization of the Kullback-Leibler (KL) divergence of the distribution P (in the original space) from the distribution Q (in the low-dimensional space) with respect to the points $\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_n$.

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

* ***early_exaggeration*** is to multiply all p_{ij} by a certain value in the initial stages of the optimization.

van der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. Journal of Machine Learning Research, 9, 2579-2605.

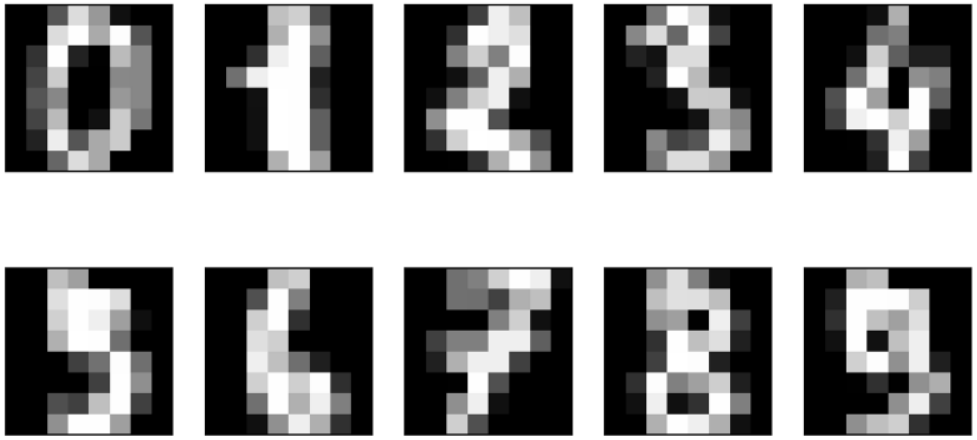
scikit-learn Practice

- **Example with the *digits* dataset**
  - The dataset consists of 1797 data points
  - Each data point in this dataset is an 8×8 grayscale image of a handwritten digit between 0 and 9.

[1]:
```
from sklearn.datasets import load_digits

digits = load_digits()
print('input shape', digits.data.shape)
print('output shape', digits.target.shape)
```

```
input shape (1797, 64)
output shape (1797,)
```

  - *Example images from the digits dataset*



scikit-learn Practice: *PCA*

- **Example (*digits* dataset)**

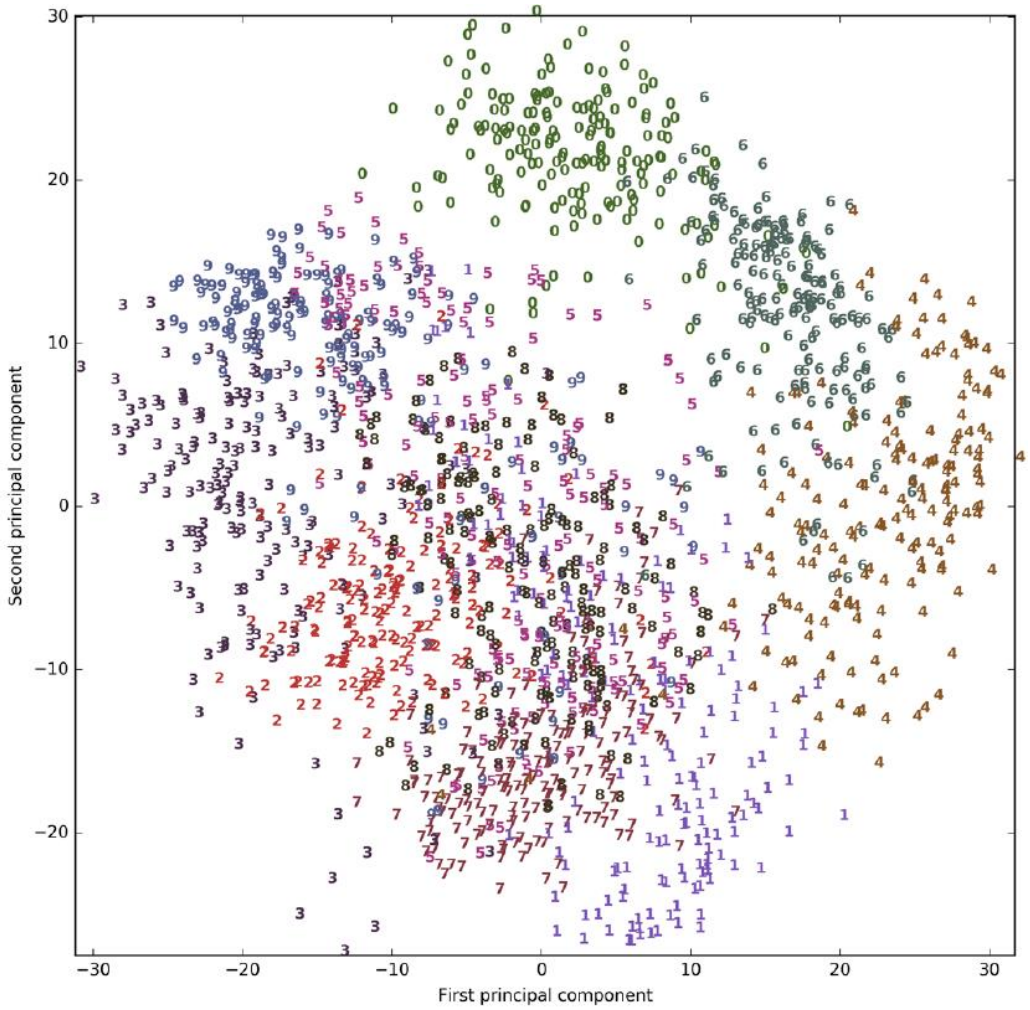
[1]:
```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
%matplotlib inline

digits = load_digits()
```

[2]:
```
pca = PCA(n_components=2)
digits_pca = pca.fit_transform(digits.data)
```

[3]:
```
colors = ['#476A2A', '#7851B8', '#BD3430', '#4A2D4E', '#875525',
          '#A83683', '#4E655E', '#853541', '#3A3120', '#535D8E']

plt.figure(figsize=(10, 10))
plt.xlim(digits_pca[:, 0].min(), digits_pca[:, 0].max())
plt.ylim(digits_pca[:, 1].min(), digits_pca[:, 1].max())

for i in range(len(digits.data)):
    # actually plot the digits as text instead of using scatter
    plt.text(digits_pca[i, 0], digits_pca[i, 1], str(digits.target[i]),
    color = colors[digits.target[i]],
    fontdict={'weight': 'bold', 'size': 9})

plt.xlabel('First principal component')
plt.ylabel('Second principal component')
```

scikit-learn Practice: *PCA*

- **Example (*digits* dataset)**
  - *Scatter plot of the digits dataset using the first two principal components*



scikit-learn Practice: *TSNE*

https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

```
class sklearn.manifold.TSNE(n_components=2, *, perplexity=30.0, early_exaggeration=12.0,
learning_rate='auto', max_iter=None, n_iter_without_progress=300, min_grad_norm=1e-07,
metric='euclidean', metric_params=None, init='pca', verbose=0, random_state=None,
method='barnes_hut', angle=0.5, n_jobs=None, n_iter='deprecated')
```

T-distributed Stochastic Neighbor Embedding.

**** It implements t-SNE with the Barnes-Hut approximation by default.***

t-SNE [1] is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.

It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples. For more tips see Laurens van der Maaten's FAQ [2].

[1] van der Maaten, L.J.P.; Hinton, G.E. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9:2579-2605, 2008.
[2] van der Maaten, L.J.P. t-Distributed Stochastic Neighbor Embedding. https://lvdmaaten.github.io/tsne/

- The same scale should be used over all features.
- The t-SNE algorithm often works well with the default settings.
- You can try playing with *perplexity* and *early_exaggeration*, but the effects are usually minor.

scikit-learn Practice: *TSNE*

https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

n_components	*int, default=2* Dimension of the embedded space.
perplexity	*float, default=30.0* The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. Different values can result in significantly different results. The perplexity must be less than the number of samples.
early_exaggeration	*float, default=12.0* Controls how tight natural clusters in the original space are in the embedded space and how much space will be between them. For larger values, the space between natural clusters will be larger in the embedded space. Again, the choice of this parameter is not very critical. If the cost function increases during initial optimization, the early exaggeration factor or the learning rate might be too high.
metric	*str or callable, default='euclidean'* The metric to use when calculating distance between instances in a feature array. If metric is a string, it must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter, or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS. If metric is "precomputed", X is assumed to be a distance matrix. Alternatively, if metric is a callable function, it is called on each pair of instances (rows) and the resulting value recorded. The callable should take two arrays from X as input and return a value indicating the distance between them. **The default is "euclidean" which is interpreted as squared euclidean distance.**

scikit-learn Practice: *TSNE*

https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

Methods	
fit(*X*, *y*=*None*)	Fit X into an embedded space.
fit_transform(*X*, *y*=*None*)	Fit X into an embedded space and return that transformed output.

* the TSNE class has no transform method
  (t-SNE does not support transforming new data)

scikit-learn Practice: *TSNE*

- **Example (*digits* dataset)**

[1]:
```
from sklearn.datasets import load_digits
from klearn.manifold import TSNE
import matplotlib.pyplot as plt
%matplotlib inline

digits = load_digits()
```

[2]:
```
tsne = TSNE(random_state=42)
digits_tsne = tsne.fit_transform(digits.data)
```

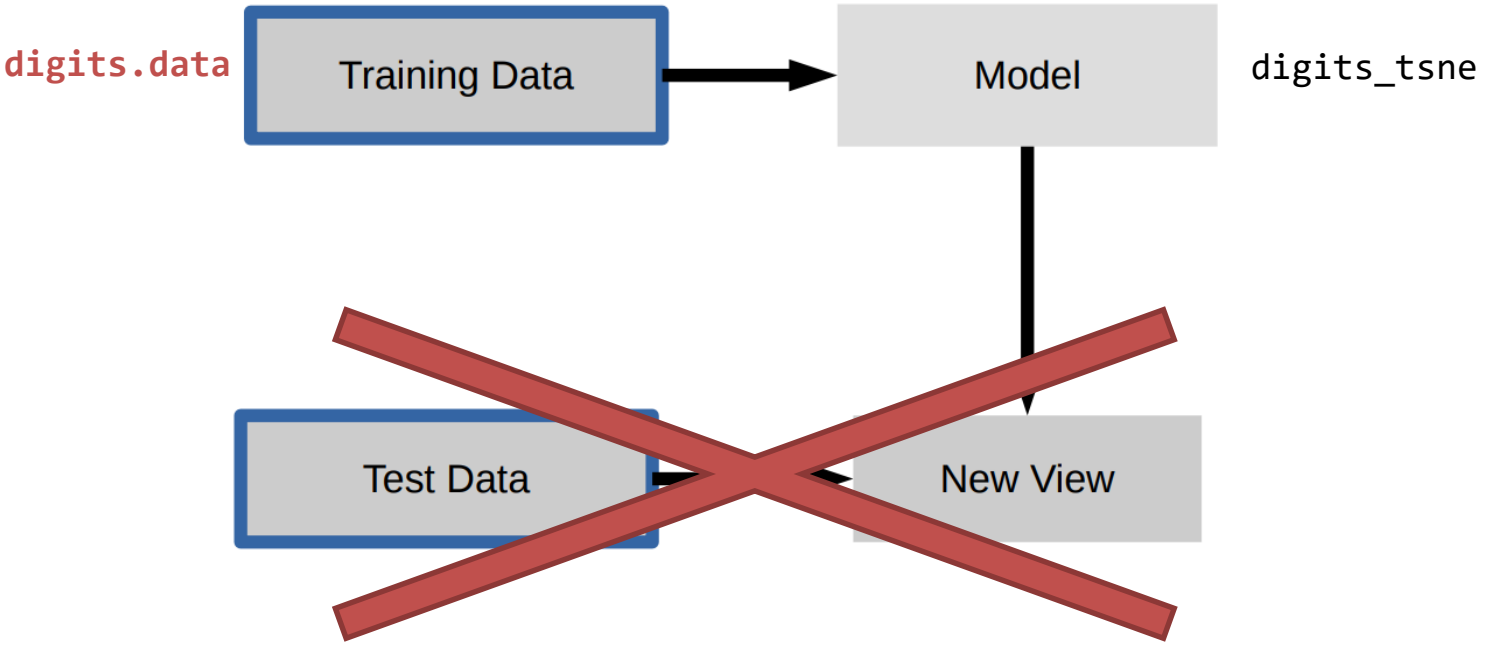
[3]:
```
colors = ['#476A2A', '#7851B8', '#BD3430', '#4A2D4E', '#875525',
          '#A83683', '#4E655E', '#853541', '#3A3120', '#535D8E']

plt.figure(figsize=(10, 10))
plt.xlim(digits_tsne[:, 0].min(), digits_pca[:, 0].max())
plt.ylim(digits_tsne[:, 1].min(), digits_pca[:, 1].max())

for i in range(len(digits.data)):
    # actually plot the digits as text instead of using scatter
    plt.text(digits_pca[i, 0], digits_pca[i, 1], str(digits.target[i]),
    color = colors[digits.target[i]],
    fontdict={'weight': 'bold', 'size': 9})

plt.xlabel('t-SNE feature 1')
plt.ylabel('t-SNE feature 2')
```

scikit-learn Practice: *TSNE*

- **Example (*digits* dataset)**

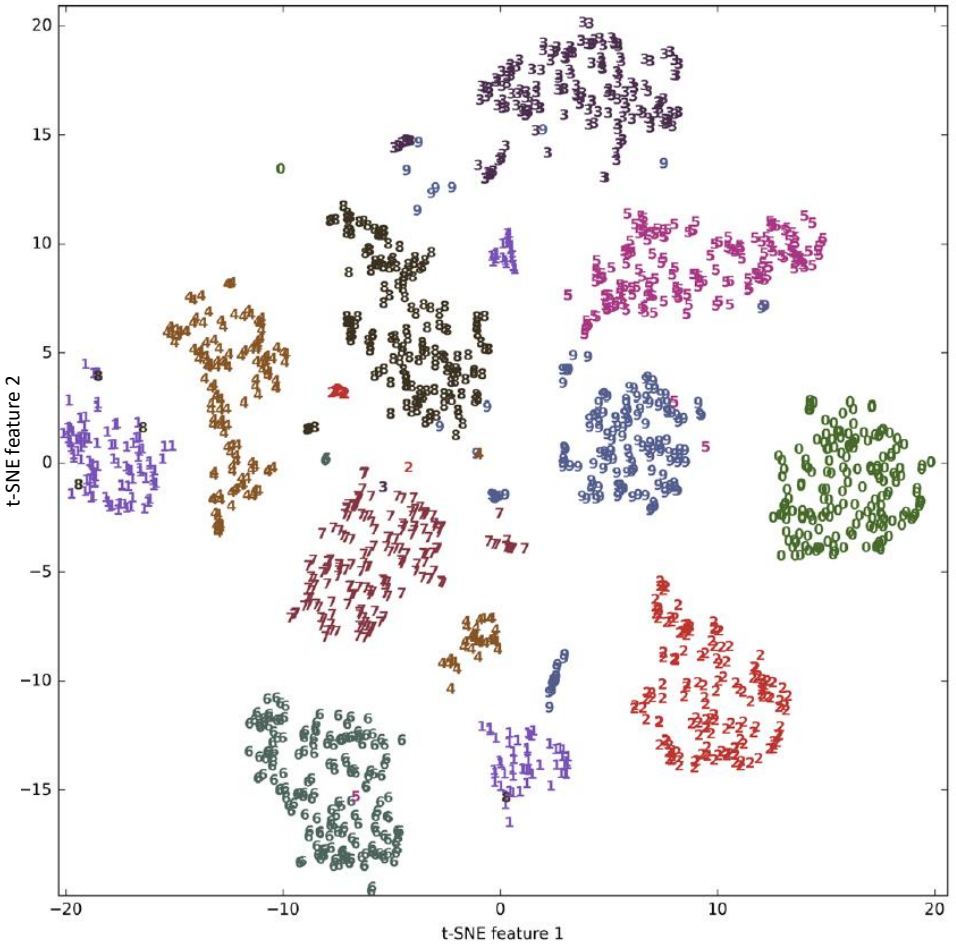
```
tsne = TSNE(random_state=42)
```

```
digits_tsne=tsne.fit_transform(digits.data)
```



scikit-learn Practice: *TSNE*

- **Example (*digits* dataset)**
  - *Scatter plot of the digits dataset using two components found by t-SNE*
    - All the classes are quite clearly separated.



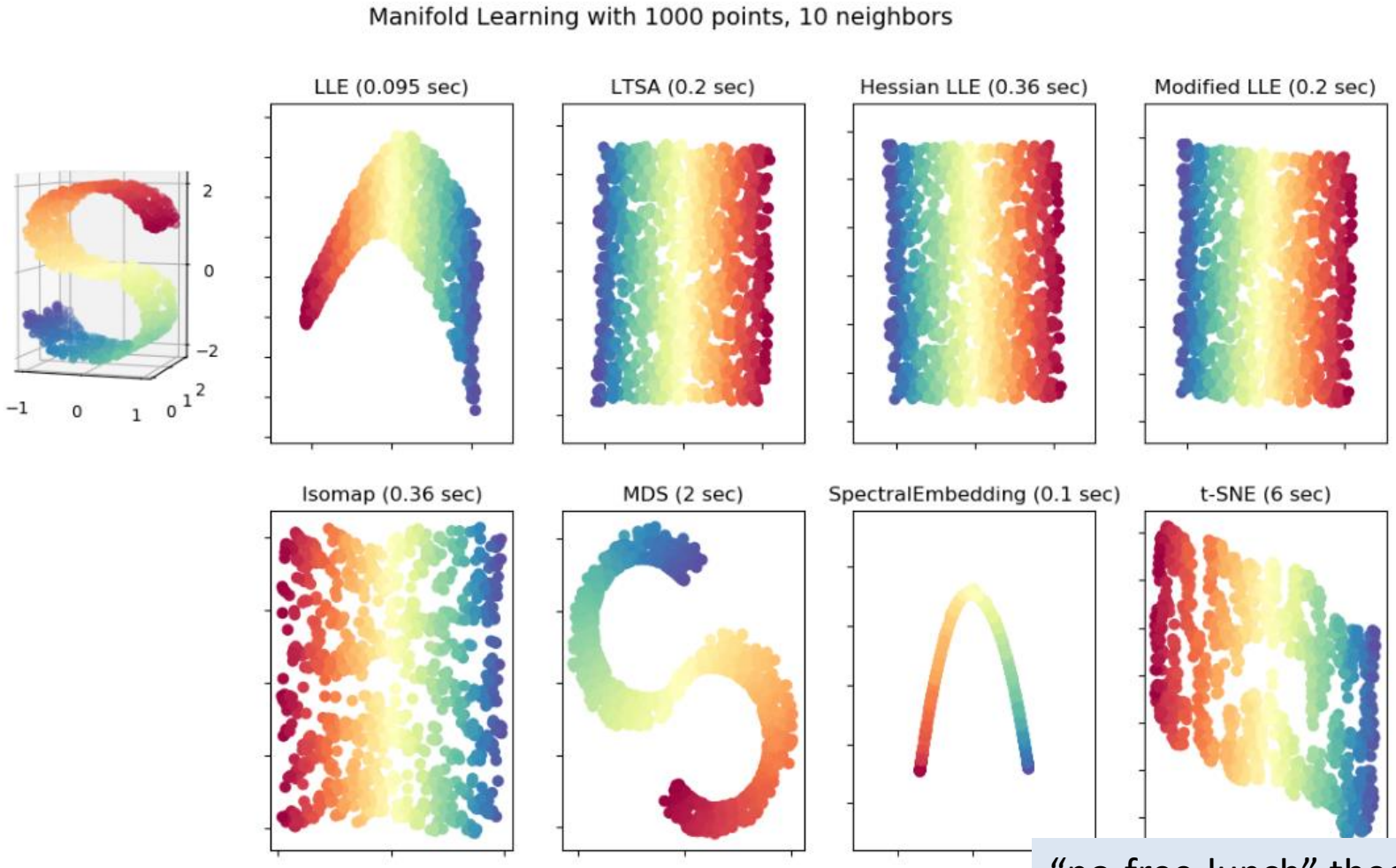
Other Dimensionality Reduction Techniques

- There are many other dimensionality reduction techniques, several of which are available in scikit-learn.
  - *Multidimensional Scaling* (MDS) reduces dimensionality while trying to preserve the distances between the data points
  - *Isomap* creates a graph by connecting each data point to its nearest neighbors, then reduces dimensionality while trying to preserve the *geodesic distances* between the data points.
  - *Locally Linear Embedding (LLE)* seeks a lower-dimensional projection of the data which preserves distances within local neighborhoods.

Other Dimensionality Reduction Techniques

https://scikit-learn.org/stable/auto_examples/manifold/plot_compare_methods.html

***Example**: Reducing the Swiss roll to 2D using* various manifold learning methods



"no-free-lunch" theorem