1. t-Distributed Stochastic Neighboring Embedding Algorithm(t-SNE)

- Non linear dimensionality reduction technique
- Visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map
- Particularly important for high-dimensional data that lie on several different, but related, low-dimensional manifolds, such as images of objects from multiple classes seen from multiple viewpoints
- t-SNE is capable of capturing much of the local structure of the high-dimensional data very well, while also revealing global structure such as the presence of clusters at several scales

1.1 Procedure

- Converts the high-dimensional Euclidean distances between datapoints into conditional probabilities that represent similarities
- The similarity of datapoint x_j to datapoint x_i is the conditional probability, p_{ji}, that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at xi.
- For nearby datapoints, p_{ji} is relatively high, whereas for widely separated datapoints, p_{ji} will be almost infinitesimal (for reasonable values of the variance of the Gaussian, σ_i).

$$P_{ij} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2/2\sigma^2)}$$

- For the low-dimensional counterparts y_i and y_j of the high-dimensional datapoints x_i and x_j, it is possible to compute a similar conditional probability, which we denote by q_{ji}
- t-SNE employs a Student t-distribution with one degree of freedom as the heavy-tailed distribution in the low-dimensional map

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

- If the map points y_i and y_j correctly model the similarity between the high-dimensional datapoints x_i and x_j, the conditional probabilities p_{ij} and q_{ij} will be equal.

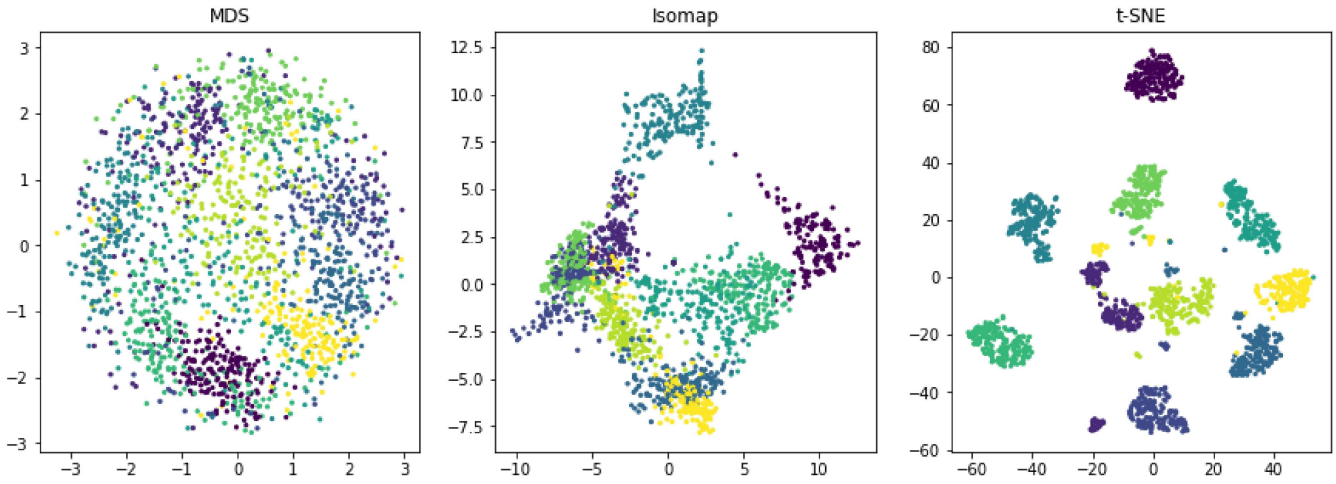
- t-SNE aims to find a low-dimensional data representation that minimizes the mismatch between p_{ij} and q_{ij}.
- t-SNE minimizes the sum of Kullback-Leibler divergences over all datapoints using a gradient descent method. The cost function C is given by:

$$J(Y) = \sum_i \sum_{j \neq i} p_{ij} \ln \frac{p_{ij}}{q_{ij}}$$

The gradient of the Kullback-Leibler divergence between P and the Student-t based joint probability distribution Q is expressed as:

$$\frac{\partial J}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

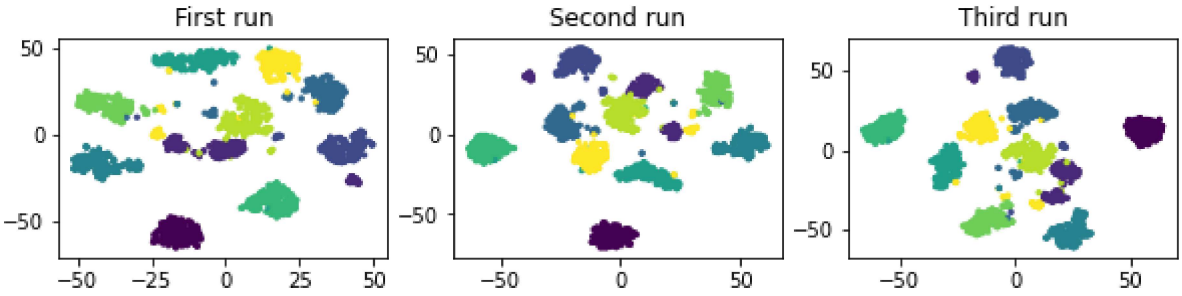
```
import numpy as np
from sklearn.manifold import Isomap
from sklearn.manifold import MDS
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

digits,classes = load_digits(return_X_y=True)
X_orig = np.array(digits.data)
y = classes
X = (X_orig - X_orig.min(axis=0))/(X_orig.max() - X_orig.min())

X_mds = MDS(n_components=2, n_init=1, max_iter=100).fit_transform(X)
X_iso = Isomap(n_components=2).fit_transform(X)
X_tsne = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0, learning_rate=200.0,

fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].scatter(X_mds[:, 0], X_mds[:, 1], 5, c=y)
axes[0].set_title('MDS')
axes[1].scatter(X_iso[:, 0], X_iso[:, 1], 5,  c=y)
axes[1].set_title('Isomap')
axes[2].scatter(X_tsne[:, 0], X_tsne[:, 1], 5, c=y)
axes[2].set_title('t-SNE')

plt.show()
```



1.2. Key points

1.2.1 Different layouts for each run

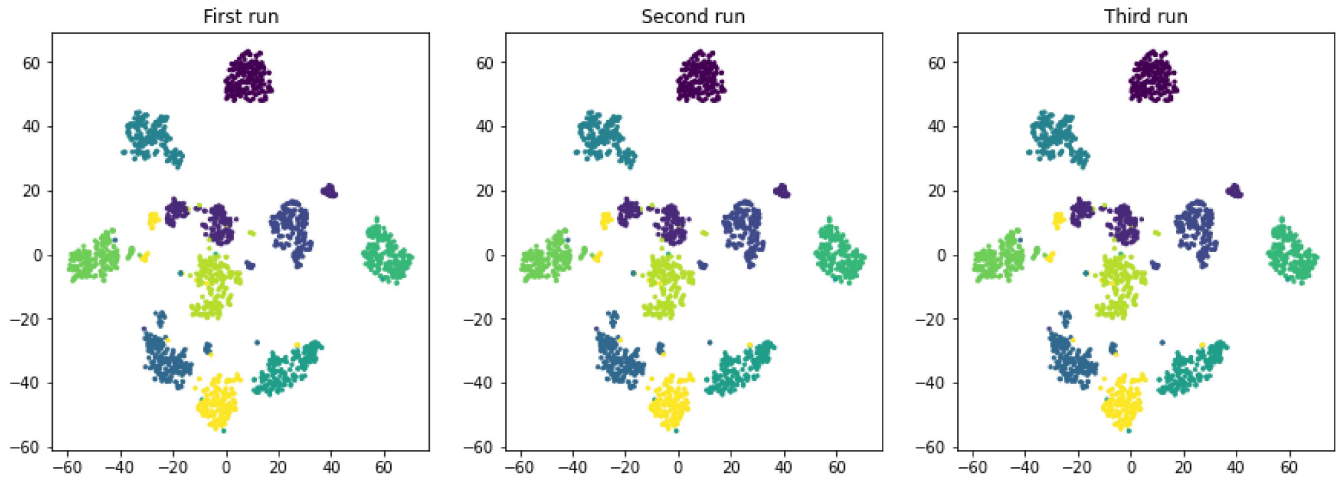
```
first_run = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0, learning_rate=200.
second_run = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0, learning_rate=200
third_run = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0, learning_rate=200.

fig, axes = plt.subplots(1, 3, figsize=(10, 2))
axes[0].scatter(first_run[:, 0], first_run[:, 1], 5, c=y)
axes[0].set_title('First run')
axes[1].scatter(second_run[:, 0], second_run[:, 1], 5, c=y)
axes[1].set_title('Second run')
axes[2].scatter(third_run[:, 0], third_run[:, 1], 5, c=y)
axes[2].set_title('Third run')

plt.show()
```



- Reasons:
  - At the beginning, t-SNE initializes the low dimension Y values randomly. That's why we may get different layout in every run
- Solution:

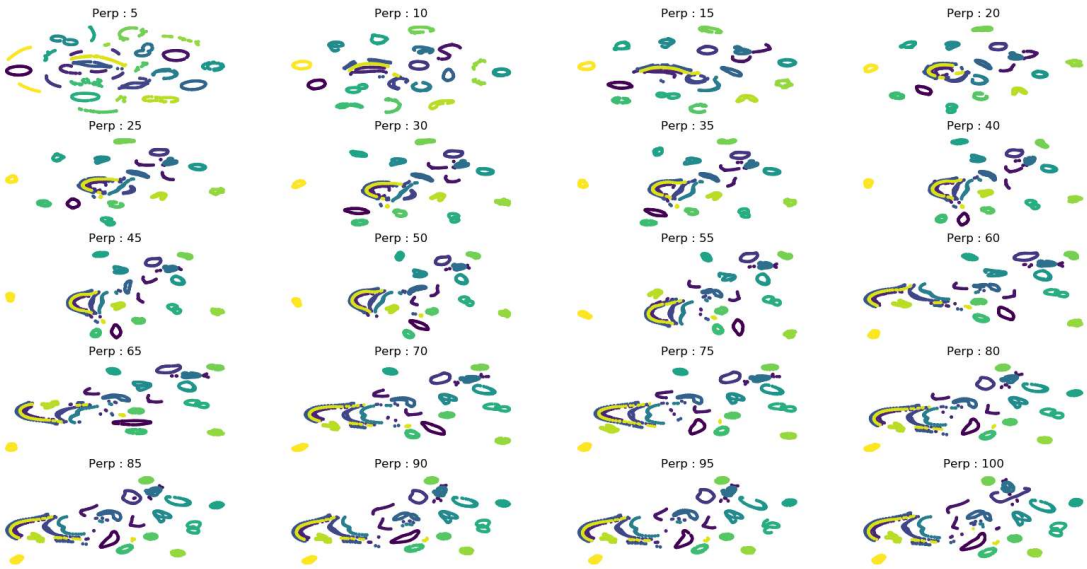
- We can fix the same the random seed to get the robust output

```
first_run = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0, learning_rate=200.
second_run = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0, learning_rate=200
third_run = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0, learning_rate=200.

fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].scatter(first_run[:, 0], first_run[:, 1], 5, c=y)
axes[0].set_title('First run')
axes[1].scatter(second_run[:, 0], second_run[:, 1], 5, c=y)
axes[1].set_title('Second run')
axes[2].scatter(third_run[:, 0], third_run[:, 1], 5, c=y)
axes[2].set_title('Third run')

plt.show()
```



1.2.2 Different perplexity can produce different layouts

The following is the experimental resuls of Columbia University Image Library(COIL-20) dataset(https://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php). This dataset consists of images of 20 different objects.

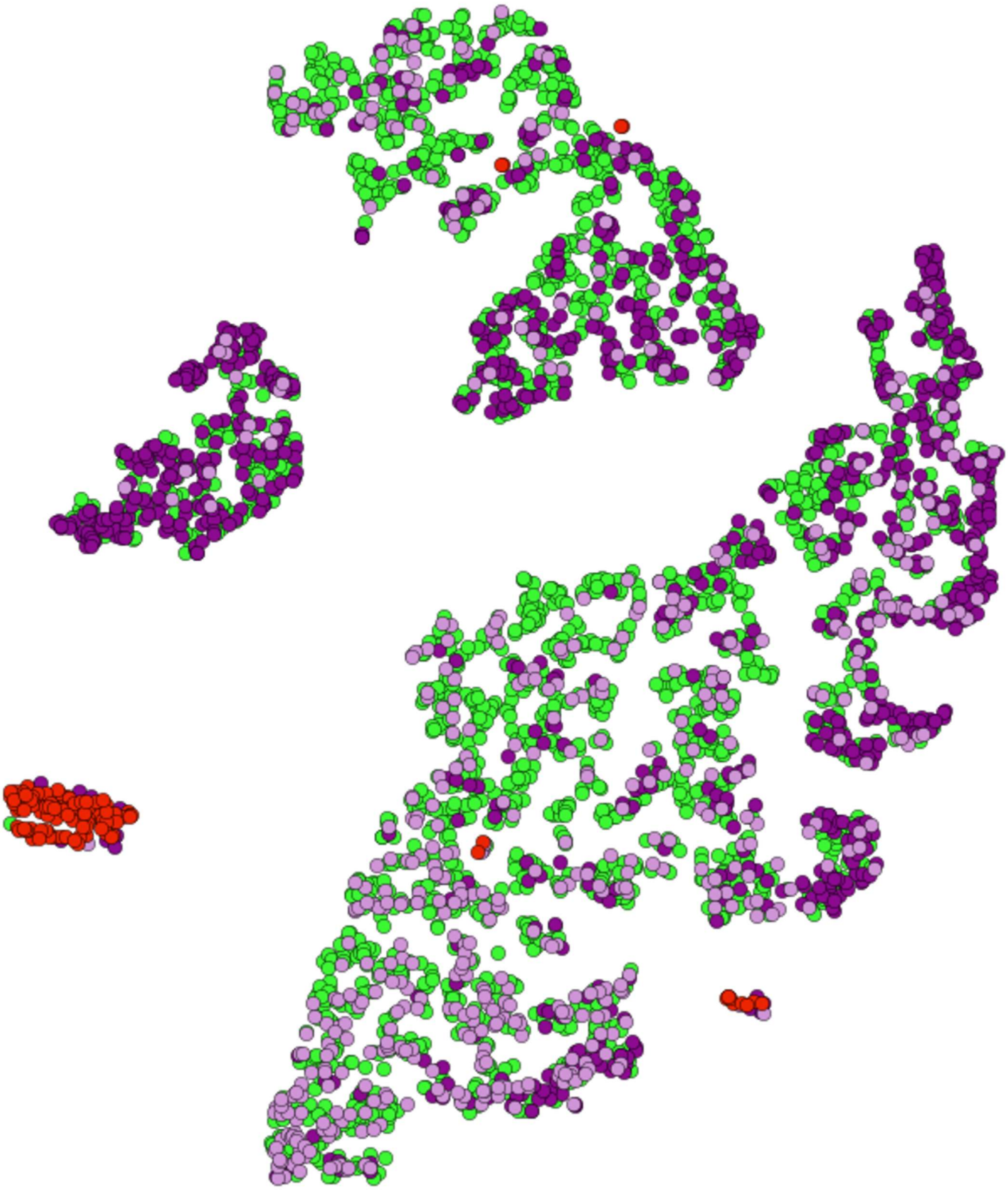


1.3 Biomedical application

1.3.1 Outlier detection(fMRI dataset)

The followings are the experimental results of functional magnetic resonance imaging dataset.

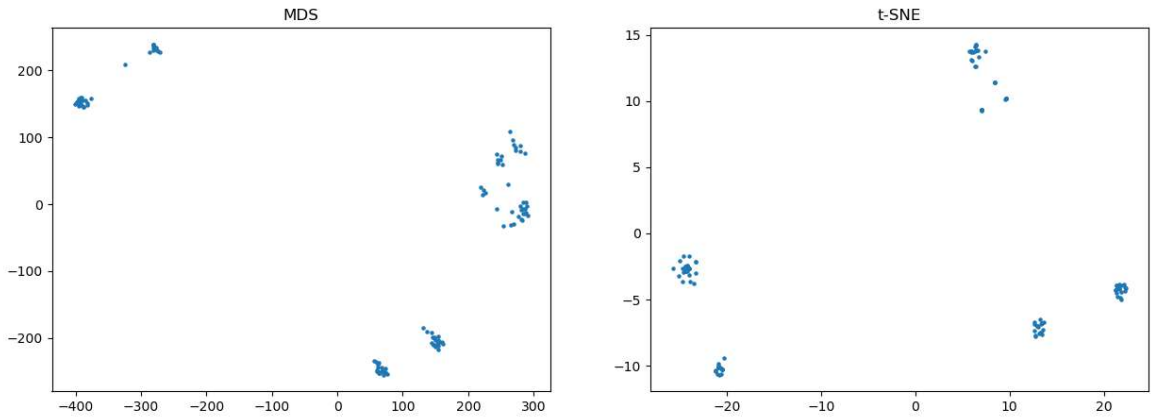
- The dataset are accumulated from multiple sites.
- One of the site contains the bad scans



Adapted from D. K. Saha et el. “doi: https://doi.org/10.1101/826974”

1.3.2 Identify transmission clusters

- The following experiment is run on a data set consisting of DNA sequences of the same length.
- Each DNA sequence is a sequence of characters from the alphabet 'A','C','T','G', and it represents a particular viral strain sampled from an infected individual.
- The pairwise distances between different DNA sequences were computed using the Hamming Distance



1.4 Observations

- Effective algorithm to embed high dimensional data to low dimensional space
- Can be used to measure the quality of data
- Need to be careful to pick the optimum perplexity
  - If the dataset are large, perplexity between 30-50 may work better
  - If the dataset are dense and small, perplexity between 10-30 may work better
- Possible to get the robust output

References

1. https://lvdmaaten.github.io/tsne/
2. https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf
3. https://github.com/deblearn