Multi-Dimension Scaling

is a distance preserving manifold learning method. All manifold learning algorithms assume the dataset lies on a smooth, non linear manifold embedded in a higher dimensional space. In MDS, the manifold is a hyperplane, and the distance between points is measured by euclidean metric. Because pairwise distance preserving, manifold learnt MDS can be modified by any distance measure of the points. It defines preserving faithful, a low dimensional embedding is achieved from the higher dimension in such a way that pairwise distances between points are preserved as much as possible. Because it preserves global structure, it is also known as global manifold learning.

MDS is a type ordination, meaning it orders MDS axes according to how much of the dataset's variance it explains. Because the dissimilarity between points (aka) joint probabilities is mapping on a lower dimension, preserving the dissimilarities as closely as possible. The dissimilarity matrix is also known as proximity matrix or distance matrix.

MDS can be divided into two categories:

- **Metric MDS**: Metric MDS is used for quantitative data and tries to preserve the original dissimilarity matrix.
- **Non-Metric MDS**: Non-metric MDS is used for ordinal data, trying to keep the order of dissimilarity matrix.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non linear technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. 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.

- The algorithm starts by calculating the probability of similarity of points in high-dimensional space and calculating the probability of similarity of points in corresponding low-dimensional space. The similarity of points is calculated as conditional probabilities that represent how much a point would choose another as its neighbor if neighbors were selected in proportion to their probability density under a Gaussian kernel centered at the first point.
- After that, it minimizes the difference between these conditional probabilities (or similarities) in higher dimensional and lower dimensional space by using Kullback-Leibler divergence of these dimensional spaces.
- To measure the minimization of the sum of difference of conditional probability t-SNE minimizes the sum of Kullback-Leibler divergences of overall data using a gradient descent method.

In simple terms, t-Distributed stochastic neighbor embedding (t-SNE) minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a similar distribution of pairwise similarities of the corresponding low-dimensional points in the embedding.

In this way, t-SNE maps the multi-dimensional data to a lower dimensional space and attempts to find patterns in the data by identifying observed clusters based on similarity of points. With small gradients, it can ensure that points have a chance to move around in the "crowd" (overlapping) and can promote clumps by inflicting repulsion only on the scale of t-SNE. Hence it is really a data exploration and visualization technique.

I specialized in bioinformatics with the amazing blogger http://pachterlab.github.io/blog/index.html, which graphically shows how uncovered cells are t-SNE and how computer science algorithms are affect RNA-seq analysis.

To better illustrate properties of individual methods of reducing dimensionality (from this and previous lab), create a set of points in the shape of a noodle. To do it:

- Create a point generator for the curve in 3-dim geometry: ensure that the radius decreases with the next step in the way you will find fit
- Train the set on a new solution with creating a sequence I mean a set
- Move the points in the set with a slight noise to avoid their perfect placement
- For better visualization, use one-pixel-thickness lines and points (depending, e.g. on the distance from the center of the noodle
- Extended neighbors should resemble the picture below

```
from time import
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import numpy
import numpy.linalg
import numpy.random
from sklearn import
from sklearn.decomposition import
from sklearn.manifold import
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.utils import check_random_state, shuffle
from scipy.optimize import minimize
from scipy.linalg import eigh
from scipy.spatial import cdist
from scipy.sparse import csr_matrix
from sklearn.utils import check_array
from sklearn.utils.extmath import _ravel
from sklearn.decomposition import _utils
from sklearn.neighbors.base import NearestNeighbors
import sklearn.neighbors.base
import sklearn.ensemble._hashing_tree_ext ifrom sklearn.ensemble._hashing_tree_ext
import sklearn.ensemble._hashing_tree_ext ifrom sklearn.ensemble._hashing_tree_ext
import sklearn.ensemble._hashing_tree_ext ifrom sklearn.ensemble._hashing_tree_ext
import sklearn.ensemble._hashing_tree_ext ifrom sklearn.ensemble._hashing_tree_ext
from sklearn.ensemble import
```

```
n = 10 # controls number of points
lengthfactor = 10
turns = 2
thetavalues = numpy.random.rand(n) * turns * 2 * numpy.pi
# for each point, how far from the center
rvalues = numpy.linspace(1, 0, n)
# add some jitter
rvalues = rvalues + (numpy.random.rand(n) - 0.5) * 0.5
rvalues = numpy.clip(rvalues, 0.0, 1.0)
# convert to cartesian
xvalues = rvalues * numpy.cos(thetavalues)
yvalues = rvalues * numpy.sin(thetavalues)
zvalues = thetavalues * lengthfactor
print("x:", xvalues)
print("y:", yvalues)
print("z:", zvalues)
```

```
x: [ 0.07 0.14 0.21 0.28 0.35 0.42 0.49 0.56 0.63 0.7 ]
y: [ 0.7 0.63 0.56 0.49 0.42 0.35 0.28 0.21 0.14 0.07]
z: [ 6.28318531 12.56637061 18.84955592 25.13274123 31.41592654 37.69911184
 43.98229715 50.26548246 56.54866776 62.83185307]
```

```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import
plt.rcParams['figure.figure_size'] = 15,6
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xvalues, yvalues, zvalues)
plt.title("Noodle")
plt.show()
```

Visualize the pairwaise set of points by projecting onto a 2D plane, changing the following aspects in subsequent approaches:

- Density of points generated, in terms of points per unit
- Amount of noise added to the points
- Dimensions reduced PCA, Kernel PCA, MDS, t-SNE
- generate at least a few plots per method

Save each generated chart, you will later need it to embedding images

PCA implementation

```
%run -d /home/jovyan/_data/pca_kernel_plot.ipynb -n -e -o -t -s -m -b -f -w -c -i -d -q -p -r
# Run if not interactive
%run -d /home/jovyan/_data/pca_kernel_plot.ipynb -n -e -o -t -s -m -b -f -w -c -i -d -q -p -r
# Run if interactive
```

```
Block: /home/jovyan/_data/pca_kernel_plot.ipynb
# If it is not interactive
# If it is not interactive
%matplotlib inline
%matplotlib inline
%matplotlib inline
```

```
plot_PCA(X[mask_digits], y[mask_digits], n_components=2, angle=(-5, 10), dataset='fc_cnn_dataset')
plot_LDA(X[mask_digits], y[mask_digits], n_components=2, angle=(-5, 10), dataset='fc_cnn_dataset')
# t-SNE takes a lot of time
# plot_TSNE(X[mask_digits], y[mask_digits], n_components=2, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_MDS(X[mask_digits], y[mask_digits], n_components=2, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_Isomap(X[mask_digits], y[mask_digits], n_components=2, angle=(-5, 10), dataset='fc_cnn_dataset')
from sklearn.manifold import LocallyLinearEmbedding
# n_components = 2
# n_neighbors = 30
# method = 'modified'
# eigen_solver = 'dense'
# X = LocallyLinearEmbedding(n_neighbors=n_neighbors,
#                            n_components=n_components,
#                            eigen_solver=eigen_solver,
#                            method=method).fit_transform(X[mask_digits])
# plot_scatter(X, y[mask_digits], angle=(-5, 10), dataset='fc_cnn_dataset')
# n_neighbors = 30
# n_components = 2
# method = 'modified'
# eigen_solver = 'dense'
# clf = LocallyLinearEmbedding(n_neighbors=n_neighbors,
#                              n_components=n_components,
#                              eigen_solver=eigen_solver,
#                              method=method)
# t0 = time.time()
# print("Computing LLE embedding")
# X_lle = clf.fit_transform(X[mask_digits])
# print("Done. Time elapsed: %f s" % (time.time() - t0))
# plot_scatter(X_lle, y[mask_digits], angle=(-5, 10), dataset='fc_cnn_dataset')
from sklearn.decomposition import SpectralEmbedding
# n_components = 2
# n_neighbors = 30
# X = SpectralEmbedding(n_components=n_components,
#                       n_neighbors=n_neighbors).fit_transform(X[mask_digits])
# plot_scatter(X, y[mask_digits], angle=(-5, 10), dataset='fc_cnn_dataset')
```

2.1.2.1

2.1.2.2

2.1.2.3

2.1.2.4

2.1.2.5

2.1.2.6

2.1.2.7

2.1.3.1

```
plot_PCA(X[mask_digits], y[mask_digits], n_components=3, angle=(-5, 10), dataset='fc_cnn_dataset')
plot_LDA(X[mask_digits], y[mask_digits], n_components=3, angle=(-5, 10), dataset='fc_cnn_dataset')
# t-SNE takes a lot of time
# plot_TSNE(X[mask_digits], y[mask_digits], n_components=3, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_MDS(X[mask_digits], y[mask_digits], n_components=3, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_Isomap(X[mask_digits], y[mask_digits], n_components=3, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_LLE(X[mask_digits], y[mask_digits], n_components=3, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_SE(X[mask_digits], y[mask_digits], n_components=3, angle=(-5, 10), dataset='fc_cnn_dataset')
```

2.1.3.2

2.1.3.3

2.1.3.4

2.1.3.5

2.1.3.6

2.1.3.7

2.1.3.8

2.1.4.1

```
plot_PCA(X[mask_digits], y[mask_digits], n_components=4, angle=(-5, 10), dataset='fc_cnn_dataset')
plot_LDA(X[mask_digits], y[mask_digits], n_components=4, angle=(-5, 10), dataset='fc_cnn_dataset')
# t-SNE takes a lot of time
# plot_TSNE(X[mask_digits], y[mask_digits], n_components=4, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_MDS(X[mask_digits], y[mask_digits], n_components=4, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_Isomap(X[mask_digits], y[mask_digits], n_components=4, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_LLE(X[mask_digits], y[mask_digits], n_components=4, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_SE(X[mask_digits], y[mask_digits], n_components=4, angle=(-5, 10), dataset='fc_cnn_dataset')
```

2.1.4.2

2.1.4.3

2.1.4.4

2.1.4.5

2.1.4.6

2.1.4.7

2.1.4.8

2.1.5

```
# t-SNE takes a lot of time
# plot_TSNE(X[mask_digits], y[mask_digits], n_components=5, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_MDS(X[mask_digits], y[mask_digits], n_components=5, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_Isomap(X[mask_digits], y[mask_digits], n_components=5, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_LLE(X[mask_digits], y[mask_digits], n_components=5, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_SE(X[mask_digits], y[mask_digits], n_components=5, angle=(-5, 10), dataset='fc_cnn_dataset')
```

2.1.5.1

2.1.5.2

2.1.5.3

2.1.5.4

2.1.5.5

2.1.5.6

2.1.5.7

2.1.6

```
# t-SNE takes a lot of time
# plot_TSNE(X[mask_digits], y[mask_digits], n_components=6, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_MDS(X[mask_digits], y[mask_digits], n_components=6, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_Isomap(X[mask_digits], y[mask_digits], n_components=6, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_LLE(X[mask_digits], y[mask_digits], n_components=6, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_SE(X[mask_digits], y[mask_digits], n_components=6, angle=(-5, 10), dataset='fc_cnn_dataset')
```

2.1.6.1

2.1.6.2

2.1.6.3

2.1.6.4

2.1.6.5

2.1.6.6

2.1.6.7

2.1.7

```
# t-SNE takes a lot of time
# plot_TSNE(X[mask_digits], y[mask_digits], n_components=7, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_MDS(X[mask_digits], y[mask_digits], n_components=7, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_Isomap(X[mask_digits], y[mask_digits], n_components=7, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_LLE(X[mask_digits], y[mask_digits], n_components=7, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_SE(X[mask_digits], y[mask_digits], n_components=7, angle=(-5, 10), dataset='fc_cnn_dataset')
```

2.1.7.1

2.1.7.2

2.1.7.3

2.1.7.4

2.1.7.5

2.1.7.6

2.1.7.7

2.1.8

```
# t-SNE takes a lot of time
# plot_TSNE(X[mask_digits], y[mask_digits], n_components=8, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_MDS(X[mask_digits], y[mask_digits], n_components=8, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_Isomap(X[mask_digits], y[mask_digits], n_components=8, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_LLE(X[mask_digits], y[mask_digits], n_components=8, angle=(-5, 10), dataset='fc_cnn_dataset')
# plot_SE(X[mask_digits], y[mask_digits], n_components=8, angle=(-5, 10), dataset='fc_cnn_dataset')
```

2.1.8.1

2.1.8.2

2.1.8.3

2.1.8.4

2.1.8.5

2.1.8.6

2.1.8.7


```
# sklearn.cross_validation import train_test_split
# from sklearn.cross_validation import cross_val_score
# from sklearn.cross_validation import Bootstrap
# from sklearn.cross_validation import KFold
# from sklearn.cross_validation import StratifiedKFold
# from sklearn.cross_validation import StratifiedShuffleSplit
# from sklearn.cross_validation import ShuffleSplit
# from sklearn.cross_validation import LeaveOneOut
# from sklearn.cross_validation import LeaveOneLabelOut
# from sklearn.cross_validation import LeavePOut
# from sklearn.cross_validation import LeavePLabelOut
# from sklearn.cross_validation import PredefinedSplit
# from sklearn.cross_validation import LabelKFold
# from sklearn.cross_validation import LabelShuffleSplit
# from sklearn.grid_search import GridSearchCV
# from sklearn.grid_search import ParameterGrid
# from sklearn.grid_search import ParameterSampler
# from sklearn.grid_search import RandomizedSearchCV
# from sklearn.preprocessing import StandardScaler
# from sklearn.preprocessing import MinMaxScaler
# from sklearn.preprocessing import Normalizer
# from sklearn.preprocessing import Binarizer
# from sklearn.preprocessing import LabelBinarizer
# from sklearn.preprocessing import MultiLabelBinarizer
# from sklearn.preprocessing import Imputer
# from sklearn.preprocessing import PolynomialFeatures
# from sklearn.preprocessing import OneHotEncoder
# from sklearn.preprocessing import FunctionTransformer
# from sklearn.feature_extraction import DictVectorizer
# from sklearn.feature_extraction import FeatureHasher
# from sklearn.feature_extraction import SkLearn
# from sklearn.feature_extraction import LabelPropagation
# from sklearn.feature_extraction import LabelSpreading
# from sklearn.feature_extraction import PassiveAggressiveClassifier
# from sklearn.feature_extraction import PassiveAggressiveRegressor
# from sklearn.feature_extraction import QuadraticDiscriminantAnalysis
# from sklearn.feature_extraction import KernelRidge
# from sklearn.feature_extraction import isotonic_regression
# from sklearn.feature_extraction import IsotonicRegression
# from sklearn.feature_extraction import sklearn_pickle_dataset
# from sklearn.feature_extraction import Scikit_Learn_Data_Sets
# from sklearn.feature_extraction import clustering_tests
# from sklearn.feature_extraction import GaussianNB
# from sklearn.feature_extraction import MultinomialNB
# from sklearn.feature_extraction import DecisionTreeClassifier
# from sklearn.feature_extraction import DecisionTreeRegressor
# from sklearn.feature_extraction import ExtraTreeClassifier
# from sklearn.feature_extraction import ExtraTreeRegressor
# from sklearn.feature_extraction import LinearSVC
# from sklearn.feature_extraction import NuSVC
# from sklearn.feature_extraction import NuSVR
# from sklearn.feature_extraction import OneClassSVM
# from sklearn.feature_extraction import SVC
# from sklearn.feature_extraction import SVR
# from sklearn.feature_extraction import AdaBoostClassifier
# from sklearn.feature_extraction import AdaBoostRegressor
# from sklearn.feature_extraction import BaggingClassifier
# from sklearn.feature_extraction import BaggingRegressor
# from sklearn.feature_extraction import ExtraTreesClassifier
# from sklearn.feature_extraction import ExtraTreesRegressor
# from sklearn.feature_extraction import GradientBoostingClassifier
# from sklearn.feature_extraction import GradientBoostingRegressor
# from sklearn.feature_extraction import RandomTreesEmbedding
# from sklearn.feature_extraction import RandomForestClassifier
# from sklearn.feature_extraction import RandomForestRegressor
# from sklearn.feature_extraction import RandomTreesEmbedding
# from sklearn.feature_extraction import VotingClassifier
# from sklearn.feature_extraction import VotingRegressor
# from sklearn.feature_extraction import Pipeline
# from sklearn.feature_extraction import FeatureUnion
# from sklearn.feature_extraction import GridSearchCV
# from sklearn.feature_extraction import ParameterGrid
# from sklearn.feature_extraction import ParameterSampler
# from sklearn.feature_extraction import RandomizedSearchCV
# from sklearn.feature_extraction import Imputer
# from sklearn.feature_extraction import MinMaxScaler
# from sklearn.feature_extraction import Normalizer
# from sklearn.feature_extraction import Binarizer
# from sklearn.feature_extraction import LabelBinarizer
# from sklearn.feature_extraction import MultiLabelBinarizer
# from sklearn.feature_extraction import PolynomialFeatures
# from sklearn.feature_extraction import OneHotEncoder
# from sklearn.feature_extraction
```

```
In [15]: mde_fitted_mnist, t_mde_mnist = MDE(verbose=1).fit_with_timing(mnist_data)
         mde_fitted_mnist.shape
         plot_2d(mde_fitted_mnist, mnist_labels, title="MDE on mnist")
         # save_image("mnist_mde.jpg", mde_fitted_mnist, mnist_labels)
         # save_image("mnist_mde.pdf", mde_fitted_mnist, mnist_labels)
```

```
/home/pynndescent/lib/python3.8/site-packages/numba/cuda/cudadrv/error.py:283: NumbaWarning: Ignoring CUDA error invalid argument (cuda.cu:780) due to missing CUDA driver.
  warnings.warn(NumbaWarning(msg))
/home/pynndescent/lib/python3.8/site-packages/numba/cuda/cudadrv/error.py:283: NumbaWarning: Ignoring CUDA error invalid argument (cuda.cu:780) due to missing CUDA driver.
  warnings.warn(NumbaWarning(msg))
/home/pynndescent/lib/python3.8/site-packages/numba/cuda/cudadrv/error.py:283: NumbaWarning: Ignoring CUDA error invalid argument (cuda.cu:780) due to missing CUDA driver.
  warnings.warn(NumbaWarning(msg))
/home/pynndescent/lib/python3.8/site-packages/numba/cuda/cudadrv/error.py:283: NumbaWarning: Ignoring CUDA error invalid argument (cuda.cu:780) due to missing CUDA driver.
  warnings.warn(NumbaWarning(msg))
```

```
In [16]: plot_2d(mde_fitted_mnist, t_sne(mnist_data), title="T_SNE on mnist")
```

PCA, kernel PCA and MDE do overlaps point from the same class, but without spacing from T-SNE (time to run) same between clusters, but results are not perfect. Due to memory constraints on MDE I couldn't use entire dataset.