TP APPC : Visualisation et réduction de dimension

Stéphane Canu

20 Novembre 2023, ASI, INSA Rouen

The purpose of this notebook is compare different data visualization algorithm to represent the MNIST data set in two dimensions order to determine

- what are the best ones to visualize data in 2 dimensions ?
- what are the scalable ones ?

The interest of using the MNIST data set it that the labels are known. We are going to apply data visualization methods without the labels and use it to asses the quality of the resulting representation[1]. Import the needed material. The function plot_repr is provided in the file `plot_repr.py`

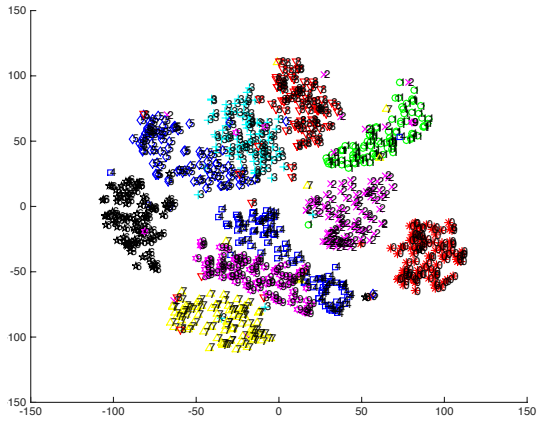

FIGURE 1 : Expected result

Ex. 1 — Dimensionality reduction and manifold learning for data visaulization

1. Data preparation.
   a) Load some relevant API

```
import numpy as np
from numpy.random import permutation
import time
import pandas as pd
import matplotlib.pyplot as plt
fw, fh = plt.rcParams["figure.figsize"]

from sklearn.datasets import fetch_openml
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.manifold import MDS, Isomap, LocallyLinearEmbedding, TSNE,
    SpectralEmbedding
from sklearn.metrics import silhouette_score, calinski_harabasz_score

from plot_repr import plot_repr
```

   b) Load MNIST data from scikit learn. Pick $n = 2000$ exemple randomly to begin with. When you will have identify the scalable data vizualization methods, you can go up to $n = 60,000$.

[1] The code is adapted from `scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html`.

```
Xini, yini = fetch_openml('mnist_784', version=1, return_X_y=True)
Xini = Xini / 255.
n = 2000
choice = permutation(Xini.shape[0])[:n]
X, y = Xini.values[choice,:], yini[choice]
```

c) Scale and normalize

```
Xc = np.float64(X - np.mean(X,0))
```

d) Choose the number of dimensions to project on

```
ndim = 2
```

After selecting the relevant ones, create a frame for the resulting metrics[2].

```
res = pd.DataFrame(index=["Computation time", "Silhouette score", "Calinski-Harabaz
    score"])
```

Why these score are relevant ?

3. PCA (principal component analysis)

a) Compute the PCA of the MNIST data in 3 different ways by using SVD, eig and sklearn[3].

```
mt = Xc.mean(axis=0)
Xf = Xc[:,np.nonzero(mt)[0]]
mt = Xf.mean(axis=0)
st = np.std(Xf,axis=0)
Xn = (Xf - mt)/st

# PCA with SVD
u, s, vh = np.linalg.svd(Xf, full_matrices=False)
print(u.shape, s.shape, vh.shape)
fig, ax = plot_repr(u[:,0:2]*s[0:2], y, title="PCA with SVD")
plt.show()

# PCA with eig
lam, v = np.linalg.eig(Xf.T@Xf)
fig, ax = plot_repr(Xf@v[:,0:2], y, title="PCA with eig")
plt.show()

# PCA with sklearn
t = time.time()
Xr = PCA(n_components=ndim).fit_transform(Xc) # eigenvalues ???
t = time.time() - t
fig, ax = plot_repr(Xr, y, title="PCA with sklearn")

res.loc[:, ax.get_title()] = (t, silhouette_score(Xr, y), calinski_harabasz_score(Xr
    , y))
plt.show()
```

b) Linear Discriminant Analysis do use the label so it should be compered with other methods. In this comparizon framework, LDA play the role of a gold standart[4] sklearn.discriminant_analysis.LinearDiscriminantAnalysis

```
t = time.time()
Xr = LinearDiscriminantAnalysis(n_components=2).fit_transform(Xc, y)
t = time.time() - t
fig, ax = plot_repr(Xr, y, title="LDA")
```

[2] http://scikit-learn.org/stable/modules/clustering.html

[3] http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[4] http://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

```
res.loc[:, ax.get_title()] = (t, silhouette_score(Xr, y), calinski_harabasz_score(Xr
    , y))
plt.show()
```

4. try now with Multidimensionnal Scaling (MDS)[5].

```
t = time.time()
Xr = MDS(n_components=ndim, metric=True).fit_transform(X)
t = time.time() - t
fig, ax = plot_repr(Xr, y, title="MDS")
res.loc[:, ax.get_title()] = (t, silhouette_score(Xr, y), calinski_harabasz_score(Xr, y)
    )
plt.show()
```

5. Isomap[6]

```
t = time.time()
Xr = Isomap(n_components=ndim, n_neighbors=10).fit_transform(X)
t = time.time() - t
fig, ax = plot_repr(Xr, y, title="Isomap")
res.loc[:, ax.get_title()] = (t, silhouette_score(Xr, y), calinski_harabasz_score(Xr, y)
    )
plt.show()
```

6. Locally Linear Embedding[7].

```
t = time.time()
Xr = LocallyLinearEmbedding(n_components=ndim, n_neighbors=10, reg=0.001).fit_transform(
    X)
t = time.time() - t
fig, ax = plot_repr(Xr, y, title="LLE")
res.loc[:, ax.get_title()] = (t, silhouette_score(Xr, y), calinski_harabasz_score(Xr, y)
    )
plt.show()
```

7. Spectral Embedding[8].

```
t = time.time()
Xr = SpectralEmbedding(n_components=ndim, affinity="nearest_neighbors", gamma=None).
    fit_transform(X)
t = time.time() - t
fig, ax = plot_repr(Xr, y, title="SE")
res.loc[:, ax.get_title()] = (t, silhouette_score(Xr, y), calinski_harabasz_score(Xr,
     y))
plt.show()
```

8. t-Distributed Stochastic Neighbor Embedding (t-SNE)[9]
   https://github.com/oreillymedia/t-SNE-tutorial

```
t = time.time()
#Xr = TSNE(n_components=ndim, n_iter=1000, init='pca',verbose=0).fit_transform(X)
Xr = TSNE(n_components=ndim, n_iter=500, verbose=0).fit_transform(X)
t = time.time() - t
fig, ax = plot_repr(Xr, y, title="t-SNE")
res.loc[:, ax.get_title()] = (t, silhouette_score(Xr, y), calinski_harabasz_score(Xr, y)
    )
plt.show()
```

[5] http://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html
[6] http://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html
[7] http://scikit-learn.org/stable/modules/generated/sklearn.manifold.LocallyLinearEmbedding.html
[8] http://scikit-learn.org/stable/modules/generated/sklearn.manifold.SpectralEmbedding.html
[9] http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

9. Spectral Embedding[10].

```
t = time.time()
Xr = SpectralEmbedding(n_components=ndim, affinity="nearest_neighbors", gamma=None).
    fit_transform(X)
t = time.time() - t
fig, ax = plot_repr(Xr, y, title="SE")
res.loc[:, ax.get_title()] = (t, silhouette_score(Xr, y), calinski_harabasz_score(Xr, y)
    )
plt.show()
```

10. Uniform Manifold Approximation and Projection (UMAP)[11]
you may need the install umap first by using
`pip install umap-learn`

```
import umap

t = time.time()
embedding = umap.UMAP(n_neighbors=10,
min_dist=0.001,
metric='correlation').fit_transform(X)
t = time.time() - t
fig, ax = plot_repr(embedding, y, title="umap")
res.loc[:, ax.get_title()] = (t, silhouette_score(embedding, y), calinski_harabasz_score
    (embedding, y))
plt.show()
```

11. Visualize and compare the results of all these methods. What are you conclusions about their quality and scalability ?

```
plt.close("all")
fig, ax = plt.subplots(3, figsize=(fw, 3 * fh))
for i in range(len(res.index)):
  ax[i].bar(res.columns.tolist(), res.iloc[i], color=plt.cm.Set2(i))
  ax[i].axhline(color="k", linewidth=plt.rcParams["axes.linewidth"])
  ax[i].set_ylabel(res.index[i])
plt.show()
res
```

12. **Your turn :**
    a) What are the list of hyperparameter of each of these methods ?
    What ar their influence ?
    b) find out another dimensionality reduction method
    c) According to your experience, which is the best method and why ?
    d) Use and compare the 3 best methods on the COIL-20 data set[12].

```
import scipy as sp
coil = sp.io.loadmat('Donnees_coil_20')
X = coil['X']
```

describe the data and build the labels...

```
nc = 20
yc = np.zeros(nc*72)
for i in range(1,nc+1):
  yc[(i-1)*72:i*72-1] = i-1;
(n,p) = X.shape
print((n,p))
```

[10] http://scikit-learn.org/stable/modules/generated/sklearn.manifold.SpectralEmbedding.html
[11] https://umap-learn.readthedocs.io/en/latest/
[12] http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php