Ministry of Education, Culture, and Research of the Republic of Moldova

Technical University of Moldova

Department of Software Engineering and Automatics

Study Program: Software Engineering

Report

Data analysis and visualisation

Done by: Ion Dodon, IS211-M

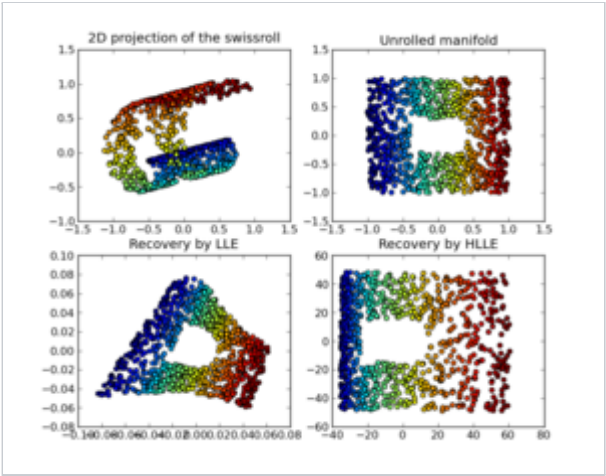
Verified by: Grozavu Nistor

Chisinau, 2021

Laboratory work no. 2

Theoretical material

Nonlinear dimensionality reduction - High-dimensional data, meaning data that requires more than two or three dimensions to represent, can be difficult to interpret. One approach to simplification is to assume that the data of interest lies within lower-dimensional space. If the data of interest is of low enough dimension, the data can be visualised in the low-dimensional space.



Top-left: a 3D dataset of 1000 points in a spiraling band (a.k.a. the Swiss roll) with a rectangular hole in the middle. Top-right: the original 2D manifold used to generate the 3D dataset. Bottom left and right: 2D recoveries of the manifold respectively using the LLE and Hessian LLE algorithms as implemented by the Modular Data Processing toolkit.

Below is a summary of some notable methods for **nonlinear dimensionality reduction**. Many of these non-linear dimensionality reduction methods are related to the linear methods listed below. Non-linear methods can be broadly classified into two groups: those that provide a mapping (either from the high-dimensional space to the low-dimensional embedding or vice versa), and those that just give a visualisation.

Locally-Linear Embedding (LLE) was presented at approximately the same time as Isomap. It has several advantages over Isomap, including faster optimization when implemented to take advantage of sparse matrix algorithms, and better results with many problems. LLE also begins by finding a set of the nearest neighbors of each point. It then computes a set of weights for each point that best describes the point as a linear

combination of its neighbors. Finally, it uses an eigenvector-based optimization technique to find the low-dimensional embedding of points, such that each point is still described with the same linear combination of its neighbors. LLE tends to handle non-uniform sample densities poorly because there is no fixed unit to prevent the weights from drifting as various regions differ in sample densities. LLE has no internal model.

Multidimensional scaling (**MDS**) is a means of visualizing the level of similarity of individual cases of a dataset. MDS is used to translate "information about the pairwise 'distances' among a set of N objects or individuals" into a configuration of N points mapped into an abstract Cartesian space.

More technically, MDS refers to a set of related ordination techniques used in information visualization, in particular to display the information contained in a distance matrix. It is a form of non-linear dimensionality reduction.

Given a distance matrix with the distances between each pair of objects in a set, and a chosen number of dimensions, *N*, an MDS algorithm places each object into *N*-dimensional space (a lower-dimensional representation) such that the between-object distances are preserved as well as possible. For $N = 1, 2,$ and 3, the resulting points can be visualized on a scatter plot.

Core theoretical contributions to MDS were made by James O. Ramsay of McGill University, who is also regarded as the founder of functional data analysis.

Conclusions

Working on this laboratory work I've user make_swiss_roll function from sklearn.datasets and plotted the data points user PCA transformation. Then I used LocallyLinearEmbedding and analyzed the resulting error for n_neighbours=[2..15]. After this was used MDS and TSNE and plotted the data to see the difference between them.

In part II I have imported digits dataset with 6 classes and different embedding like Random projection embedding, Truncated SVD embedding, and others were used to transform the dataset. Then the Decision Tree model was used to work with the projections given by each embedding method. Then I compared the error and score after fitting with each projection.

Bibliography

- https://en.wikipedia.org/wiki/Multidimensional_scaling
- https://en.wikipedia.org/wiki/Dimensionality_reduction
- https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction

LaboratoryWork2

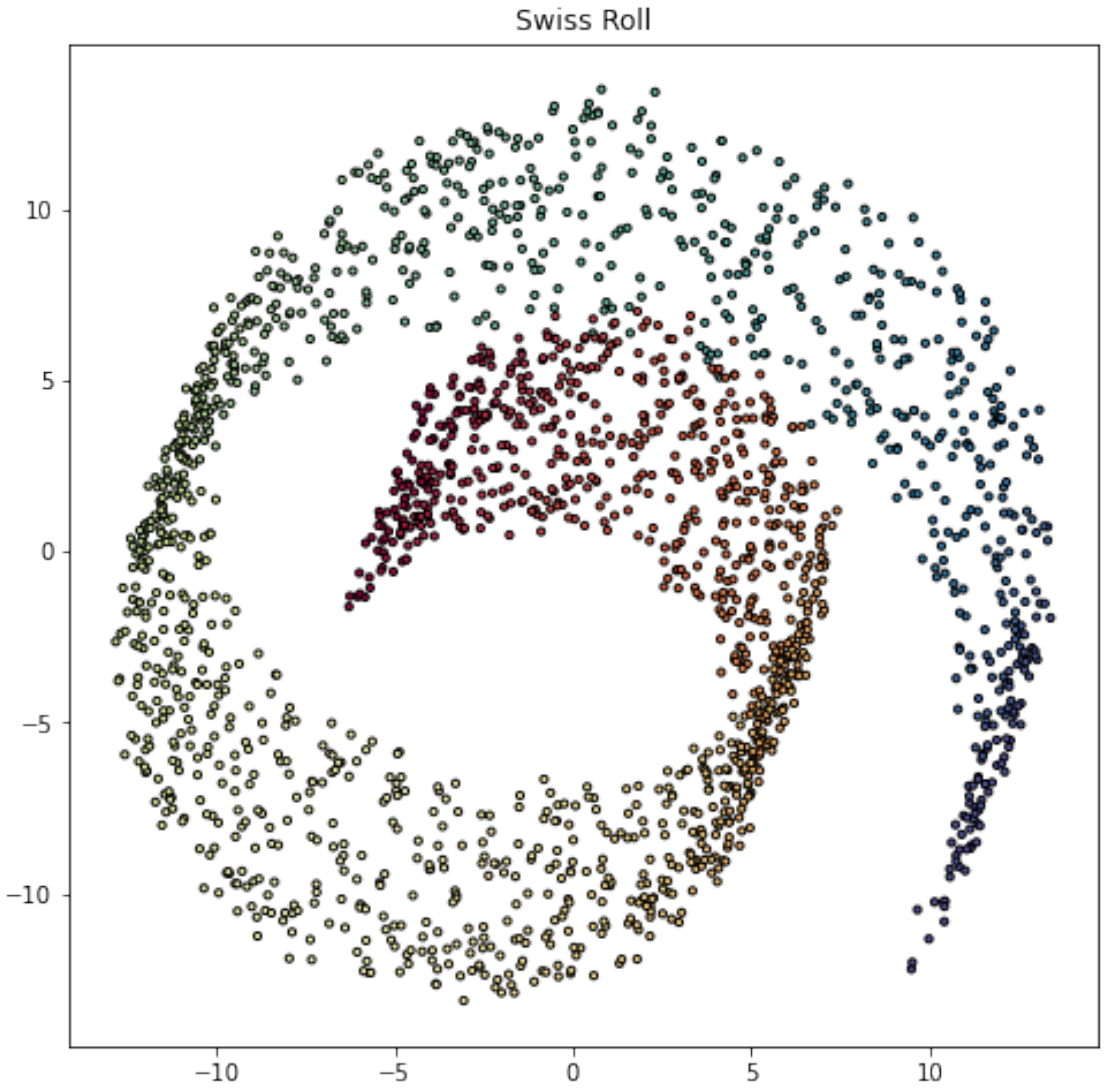
January 4, 2022

0.0.1 Part I. Swiss-roll datase

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
from sklearn.decomposition import PCA
from sklearn.manifold import LocallyLinearEmbedding
from sklearn.manifold import MDS
from sklearn.manifold import TSNE
from sklearn.datasets import load_digits
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

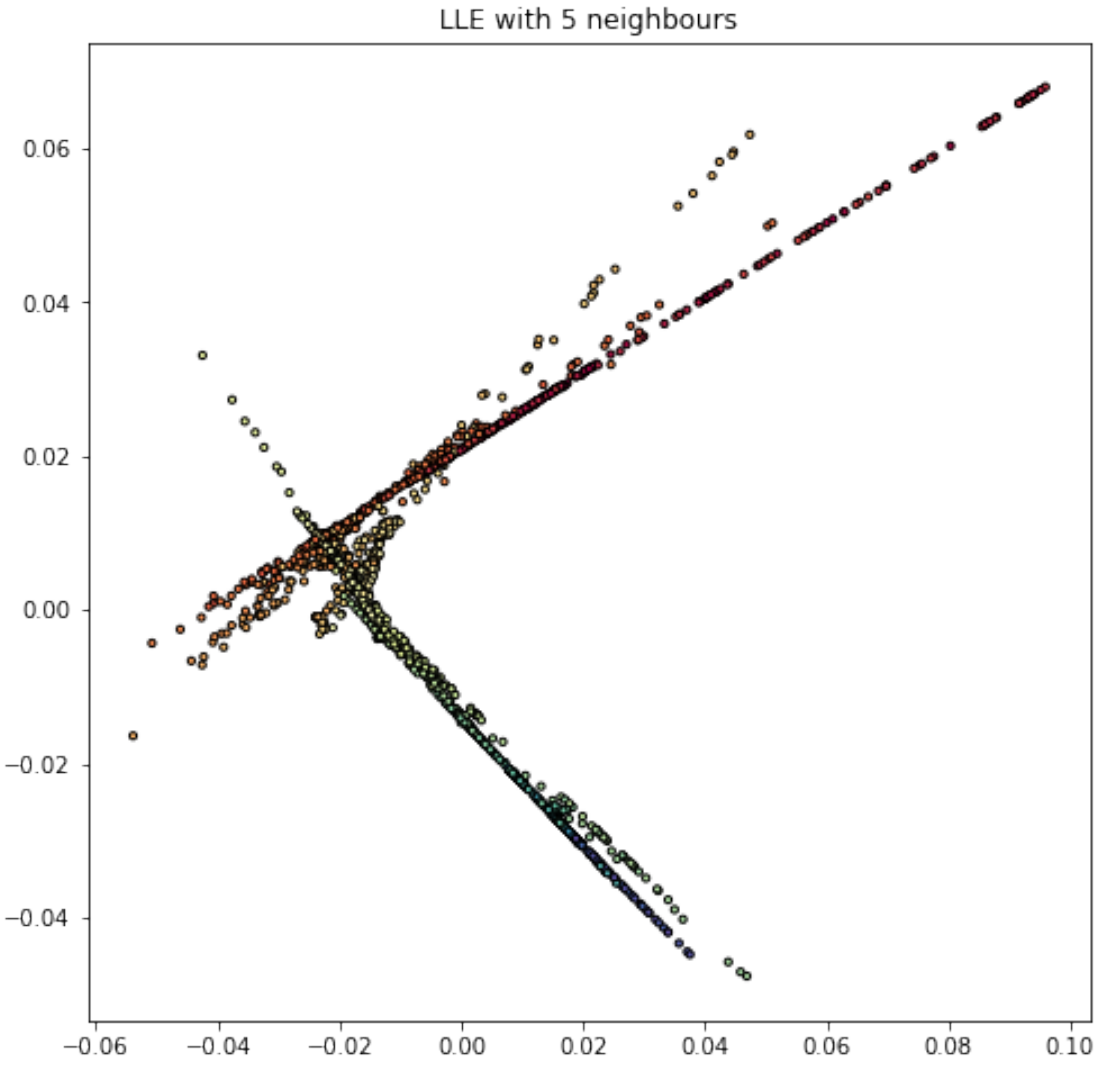
[ ]:
```
def plot_swiss_roll(X, y, title):
    """
    Plot the first 2000 points of the swis-roll dataset.
    """
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)
    plt.figure(figsize=(8, 8))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, s=10, edgecolor='k', cmap=plt.cm.
 ↪get_cmap('Spectral'))
    plt.title(title)
    plt.show()
```

[ ]:
```
# Generate the swis-roll dataset with 2000 points using the functio datasets.
 ↪make_swiss_roll
# and plot the first 100 points.

X, y = make_swiss_roll(n_samples=2000, noise=0.2)
plot_swiss_roll(X, y, "Swiss Roll")
```



[ ]:
```
def plot(X, y, title):
    """
    Plot the first 2000 points of the swis-roll dataset.
    """
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)
    plt.figure(figsize=(8, 8))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, s=10, edgecolor='k', cmap=plt.cm.
↪get_cmap('Spectral'))
    plt.title(title)
    plt.show()
```

[ ]:
```
# Apply LLE (Local Linear Embedding) with 5 neighbours by printing the error

lle = LocallyLinearEmbedding(n_neighbors=5, n_components=2)
X_lle = lle.fit_transform(X)
plot(X_lle, y, "LLE with 5 neighbours")
print(lle.reconstruction_error_)
```



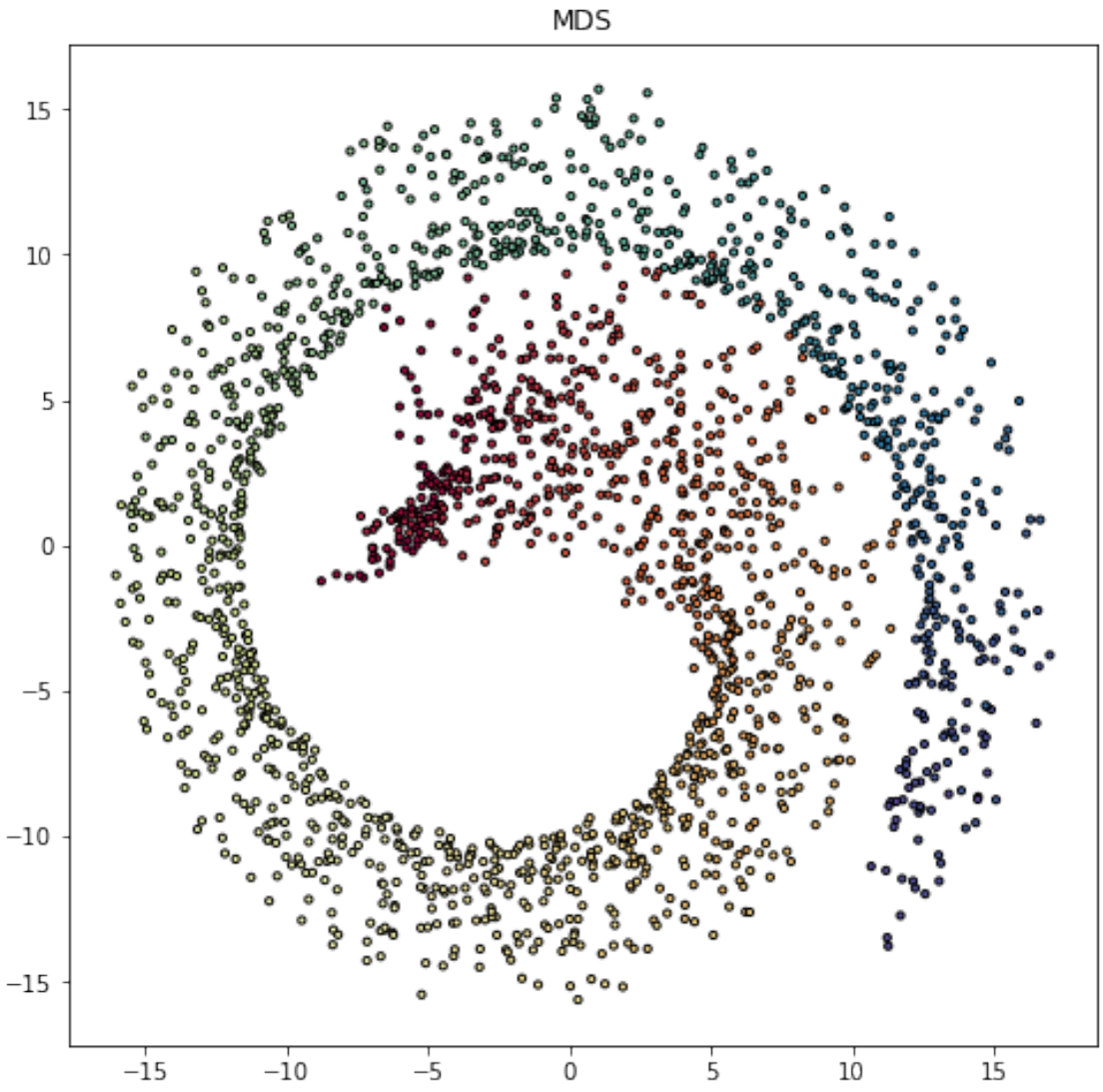

```
2.4360173452880945e-12
```

[ ]:
```
# Change the number of neighbours from 2 to 15 and plot the error line. Which␣
↪is the best number of neighbours ?

error = np.Infinity
best_neighbours = None
```

```
for n_neighbors in range(2, 15):
    lle = LocallyLinearEmbedding(n_neighbors=n_neighbors, eigen_solver='dense')
    X_lle = lle.fit_transform(X)
    if lle.reconstruction_error_ < error:
        error = lle.reconstruction_error_
        best_neighbours = n_neighbors
    print("n_neighbours = {}: Reconstruction error = {}".format(n_neighbors,␣
 ↪lle.reconstruction_error_))

print("The best number of neighbors is {}.".format(best_neighbours))
```

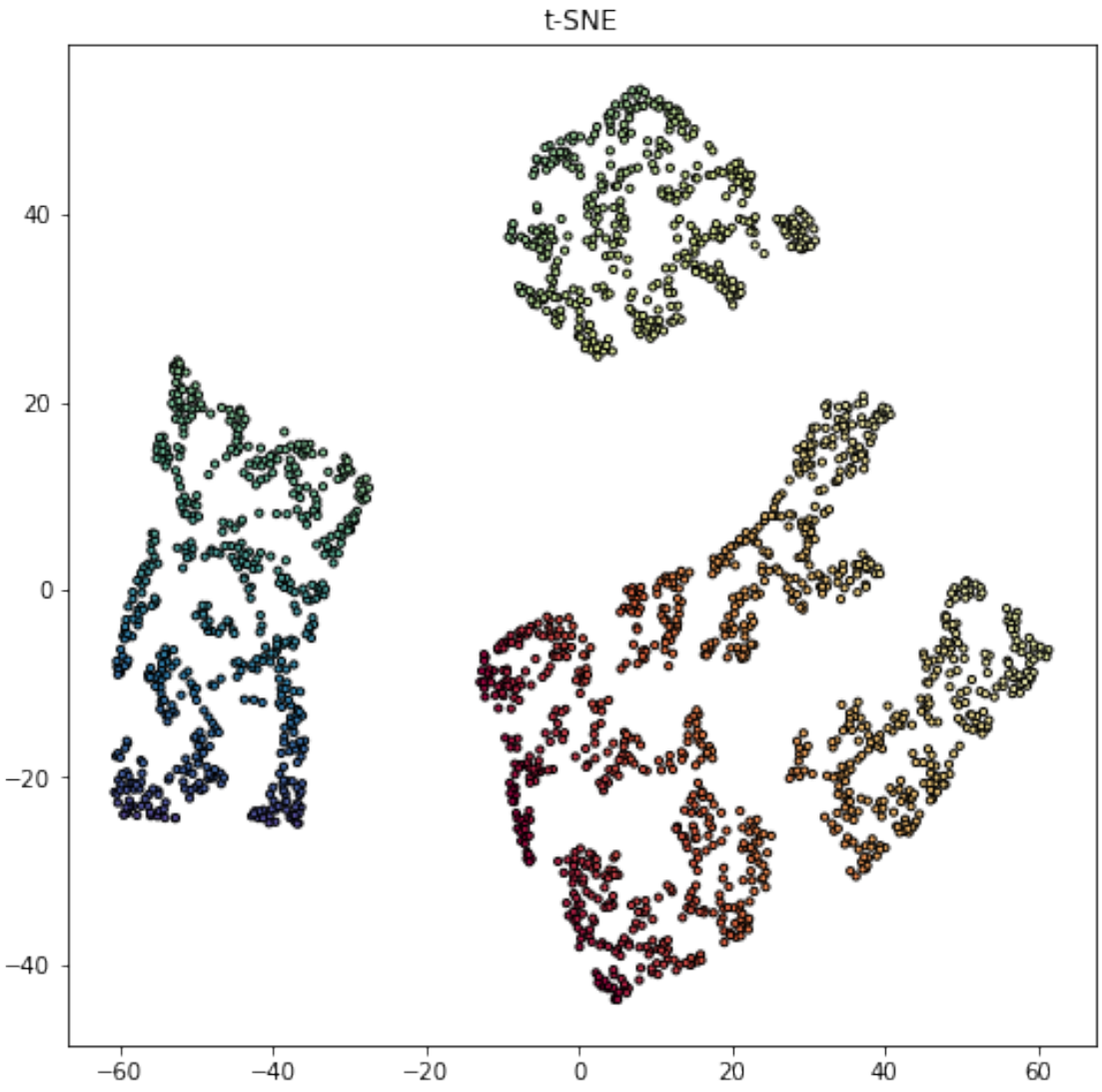
```
n_neighbours = 2: Reconstruction error = -6.289026807873649e-15
n_neighbours = 3: Reconstruction error = -1.836678988051594e-16
n_neighbours = 4: Reconstruction error = -2.611543152542782e-16
n_neighbours = 5: Reconstruction error = 2.4351417956970172e-12
n_neighbours = 6: Reconstruction error = 1.4595895512299734e-10
n_neighbours = 7: Reconstruction error = 2.5605038867055727e-10
n_neighbours = 8: Reconstruction error = 2.962623980351711e-09
n_neighbours = 9: Reconstruction error = 8.233248359376546e-10
n_neighbours = 10: Reconstruction error = 7.446463699008286e-09
n_neighbours = 11: Reconstruction error = 2.8208150443870565e-08
n_neighbours = 12: Reconstruction error = 3.525462456301093e-08
n_neighbours = 13: Reconstruction error = 3.150649230884524e-08
n_neighbours = 14: Reconstruction error = 3.242402318890731e-08
The best number of neighbors is 2.
```

[ ]:
```
# Use Multi Dimensional Scaling with manifold.MDS and visualize the dataset in␣
 ↪2 dimension

mds = MDS(n_components=2, max_iter=100, n_init=1)
X_mds = mds.fit_transform(X)
plot(X_mds, y, "MDS")
```



[ ]:
```
# Apply t-SNE model to the same dataset with manifold.TSNE

tsne = TSNE(n_components=2, init='pca', random_state=0)
X_tsne = tsne.fit_transform(X)
plot(X_tsne, y, "t-SNE")
```

```
/home/ion/.local/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:790:
FutureWarning: The default learning rate in TSNE will change from 200.0 to
'auto' in 1.2.
  warnings.warn(
/home/ion/.local/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:982:
FutureWarning: The PCA initialization in TSNE will change to have the standard
deviation of PC1 equal to 1e-4 in 1.2. This will ensure better convergence.
  warnings.warn(
```



The best model is t-SNE because it could group the data into two dimensions and create clusters that contains pointsof the same color. Even if we reexecute the code, there will be other clusters that may collide, but the model will always be able to create distiguishable clusters. With MDS the are no clusters but the model could create a spiral and separated the red points from the blue points.

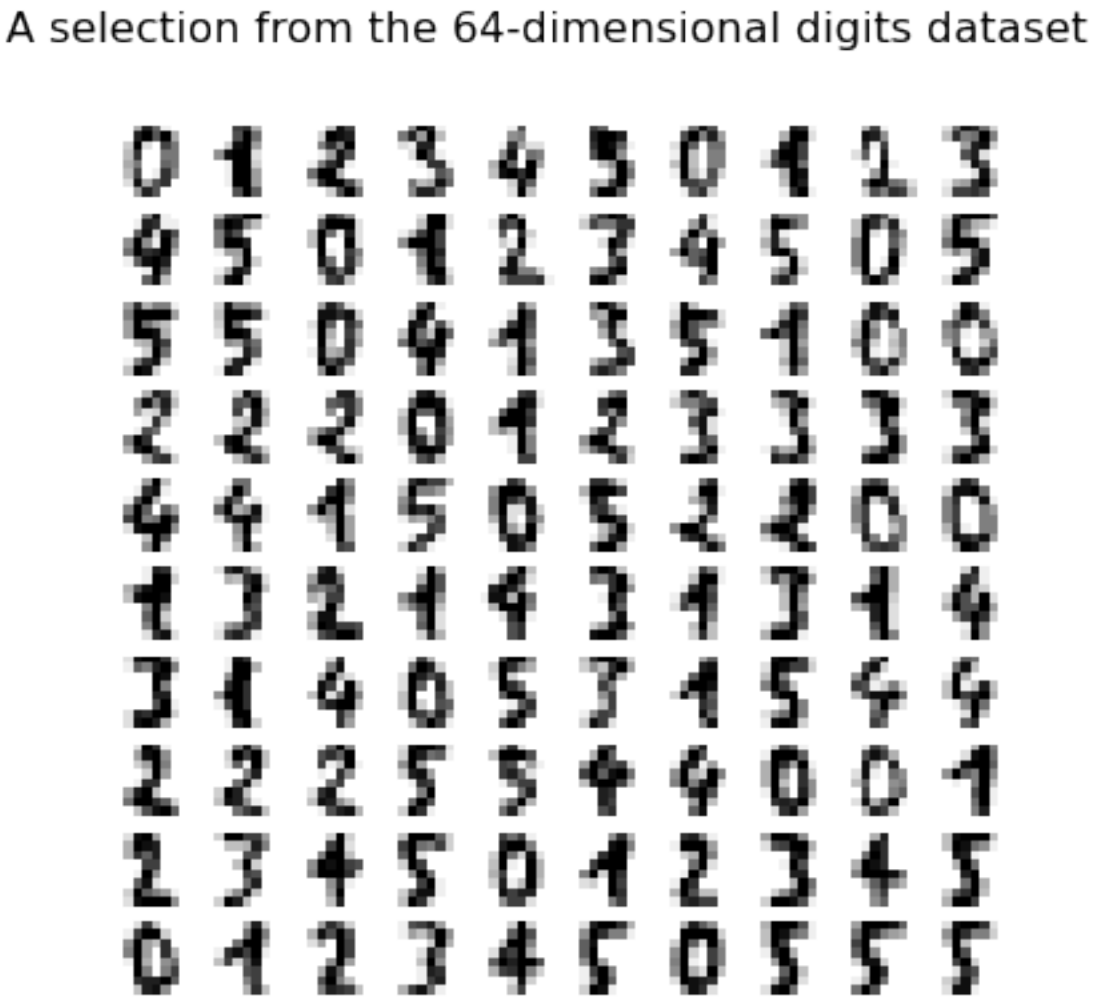
0.0.2 Part II. Digit dataset

[ ]:
```
def plot_digits(X, y, title):
    """
    Plot the first 100 digits of the dataset.
    """
    plt.figure(figsize=(8, 8))
```

```
    plt.imshow(np.reshape(X[0], (8, 8)), cmap=plt.cm.gray)
    plt.title(title)
    plt.show()
```

[ ]:
```
# Import the digit dataset containing only 6 classes

digits = load_digits(n_class=6)
X, y = digits.data, digits.target
n_samples, n_features = X.shape
```

[ ]:
```
fig, axs = plt.subplots(nrows=10, ncols=10, figsize=(6, 6))
for idx, ax in enumerate(axs.ravel()):
    ax.imshow(X[idx].reshape((8, 8)), cmap=plt.cm.binary)
    ax.axis("off")
_ = fig.suptitle("A selection from the 64-dimensional digits dataset",␣
 ↪fontsize=16)
```



[ ]:
```
import numpy as np
from matplotlib import offsetbox
from sklearn.preprocessing import MinMaxScaler

def plot_embedding(X, title, ax):
    X = MinMaxScaler().fit_transform(X)
    for digit in digits.target_names:
        ax.scatter(
            *X[y == digit].T,
            marker=f"${digit}$",
            s=60,
            color=plt.cm.Dark2(digit),
            alpha=0.425,
            zorder=2,
        )
    shown_images = np.array([[1.0, 1.0]])  # just something big
    for i in range(X.shape[0]):
        # plot every digit on the embedding
        # show an annotation box for a group of digits
        dist = np.sum((X[i] - shown_images) ** 2, 1)
        if np.min(dist) < 4e-3:
            # don't show points that are too close
            continue
        shown_images = np.concatenate([shown_images, [X[i]]], axis=0)
        imagebox = offsetbox.AnnotationBbox(
            offsetbox.OffsetImage(digits.images[i], cmap=plt.cm.gray_r), X[i]
        )
        imagebox.set(zorder=1)
        ax.add_artist(imagebox)

    ax.set_title(title)
    ax.axis("off")
```

[ ]:
```
from sklearn.decomposition import TruncatedSVD
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomTreesEmbedding
from sklearn.manifold import (
    Isomap,
    LocallyLinearEmbedding,
    MDS,
    SpectralEmbedding,
    TSNE,
)
from sklearn.neighbors import NeighborhoodComponentsAnalysis
from sklearn.pipeline import make_pipeline
from sklearn.random_projection import SparseRandomProjection
```

```
embeddings = {
    "Random projection embedding": SparseRandomProjection(
        n_components=2, random_state=42
    ),
    "Truncated SVD embedding": TruncatedSVD(n_components=2),
    "Linear Discriminant Analysis embedding": LinearDiscriminantAnalysis(
        n_components=2
    ),
    "Isomap embedding": Isomap(n_neighbors=n_neighbors, n_components=2),
    "Standard LLE embedding": LocallyLinearEmbedding(
        n_neighbors=n_neighbors, n_components=2, method="standard"
    ),
    "Modified LLE embedding": LocallyLinearEmbedding(
        n_neighbors=n_neighbors, n_components=2, method="modified"
    ),
    "Hessian LLE embedding": LocallyLinearEmbedding(
        n_neighbors=n_neighbors, n_components=2, method="hessian",␣
↪eigen_solver="dense"
    ),
    "LTSA LLE embedding": LocallyLinearEmbedding(
        n_neighbors=n_neighbors, n_components=2, method="ltsa"
    ),
    "MDS embedding": MDS(n_components=2, n_init=1, max_iter=120, n_jobs=2),
    "Random Trees embedding": make_pipeline(
        RandomTreesEmbedding(n_estimators=200, max_depth=5, random_state=0),
        TruncatedSVD(n_components=2),
    ),
    "Spectral embedding": SpectralEmbedding(
        n_components=2, random_state=0, eigen_solver="arpack"
    ),
    "t-SNE embeedding": TSNE(
        n_components=2,
        init="pca",
        learning_rate="auto",
        n_iter=500,
        n_iter_without_progress=150,
        n_jobs=2,
        random_state=0,
    ),
    "NCA embedding": NeighborhoodComponentsAnalysis(
        n_components=2, init="pca", random_state=0
    ),
}
```

[ ]:
```
from time import time

projections, timing = {}, {}
```

```
for name, transformer in embeddings.items():
    if name.startswith("Linear Discriminant Analysis"):
        data = X.copy()
        data.flat[:: X.shape[1] + 1] += 0.01  # Make X invertible
    else:
        data = X

    print(f"Computing {name}...")
    start_time = time()
    projections[name] = transformer.fit_transform(data, y)
    timing[name] = time() - start_time
```

```
Computing Random projection embedding…
Computing Truncated SVD embedding…
Computing Linear Discriminant Analysis embedding…
Computing Isomap embedding…
Computing Standard LLE embedding…
Computing Modified LLE embedding…
Computing Hessian LLE embedding…
Computing LTSA LLE embedding…
Computing MDS embedding…
```

```
/home/ion/.local/lib/python3.9/site-
packages/scipy/sparse/linalg/eigen/arpack/arpack.py:936: LinAlgWarning: Diagonal
number 350 is exactly zero. Singular matrix.
  self.M_lu = lu_factor(M)
```

```
Computing Random Trees embedding…
Computing Spectral embedding…
Computing t-SNE embeedding…
```

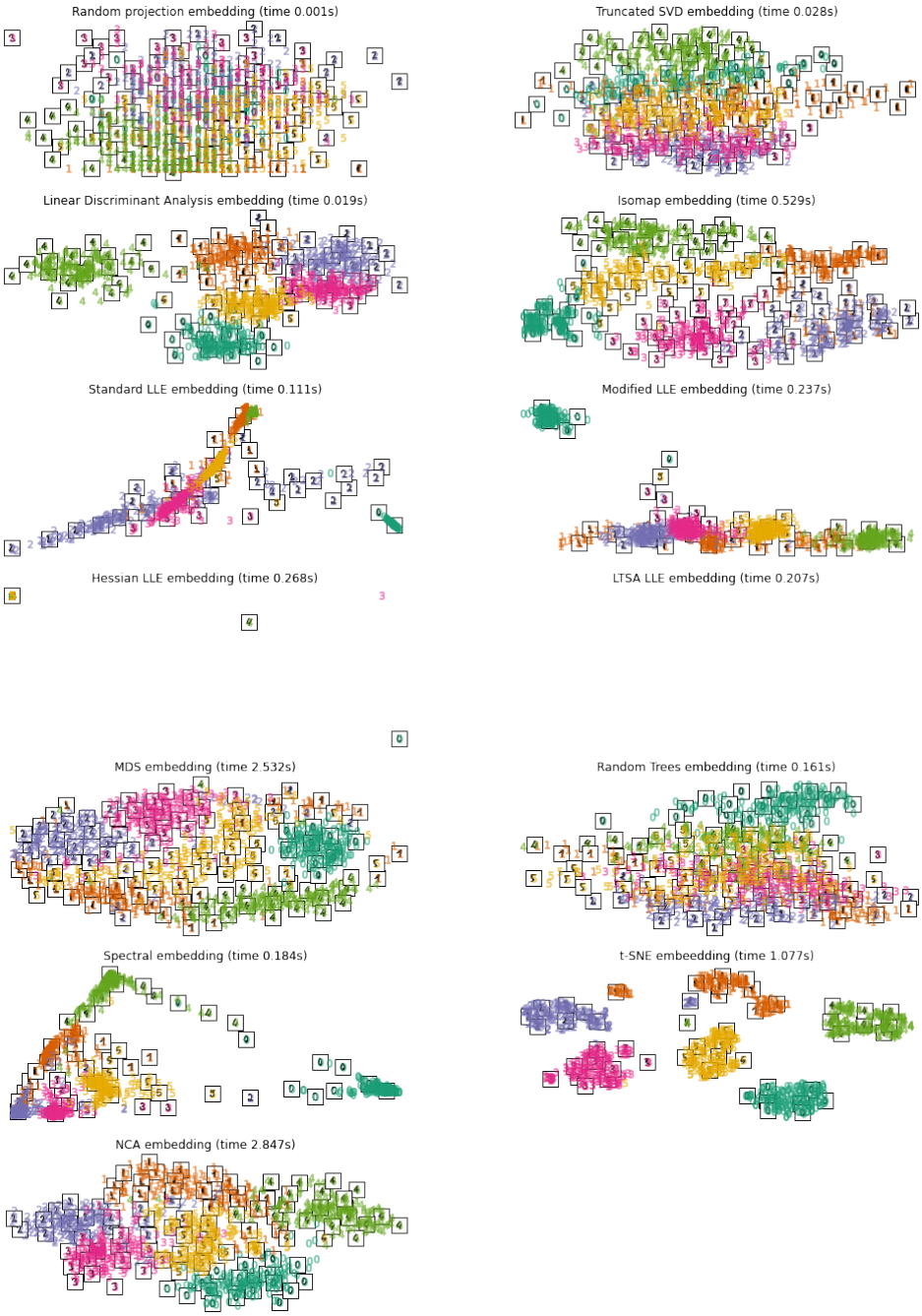
```
/home/ion/.local/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:982:
FutureWarning: The PCA initialization in TSNE will change to have the standard
deviation of PC1 equal to 1e-4 in 1.2. This will ensure better convergence.
  warnings.warn(
```

```
Computing NCA embedding…
```

[ ]:
```
from itertools import zip_longest

fig, axs = plt.subplots(nrows=7, ncols=2, figsize=(17, 24))

for name, ax in zip_longest(timing, axs.ravel()):
    if name is None:
        ax.axis("off")
        continue
    title = f"{name} (time {timing[name]:.3f}s)"
    plot_embedding(projections[name], title, ax)
```

```
plt.show()
```

```
/home/ion/.local/lib/python3.9/site-packages/sklearn/preprocessing/_data.py:461:
RuntimeWarning: All-NaN slice encountered
  data_min = np.nanmin(X, axis=0)
/home/ion/.local/lib/python3.9/site-packages/sklearn/preprocessing/_data.py:462:
RuntimeWarning: All-NaN slice encountered
  data_max = np.nanmax(X, axis=0)
```



Analyse the results by explaining the models

- The embeding that best grouped the data is t-SNE and it took more time than most of the other embeding like Isomap embedding or Linear discriminant analysis.
- Spectral embedding tends to group the clusters farther from each other and tends to make the clusters dense.
- NCA embedding took the most of the time. It did not create as dense a cluster as Spectral embedding did and the clusters a closer to each other. The results are very similar to MDS embedding which also took a similar amount of time.
- Random tree embedding did not do as a good job as the ones mentioned above and it took much less time. The points are more spread, the clusters collide, and are badly distinguishable.
- Random Projection embedding took the least of time and the points are very spread, with closer clusters that collide. A similar result was given by Truncated SVD embedding, but this one took a bit longer and gave a better result.
- Linear Discriminant Analysis embedding created clusters where the points are closer to each other in comparison to Isomap embedding. LDA is faster and I would say it did a better job than Isomap embedding.
- Hassien LLE creates clusters where the points are in the same position while LTSA LLE didn't create any clusters.
- In the Standart LLE embedding the clusters are more visible and more spread and Modified LLE embedding.

[ ]:
```
from sklearn.tree import DecisionTreeClassifier

# for each projection
for name, projection in projections.items():
    clf = DecisionTreeClassifier(random_state=0)
    try:
        clf.fit(projection, y)
    except ValueError:
        print(f"{name}: cannot fit a decision tree. Nan Values found in␣
 ↪projection.")
        continue
    score = clf.score(projection, y)
    print(f"{name}: Error = {1 - score:.3f} | Score = {score:.3f}")
```

```
Random projection embedding: Error = 0.282 | Score = 0.718
Truncated SVD embedding: Error = 0.000 | Score = 1.000
Linear Discriminant Analysis embedding: Error = 0.000 | Score = 1.000
Isomap embedding: Error = 0.000 | Score = 1.000
Standard LLE embedding: Error = 0.000 | Score = 1.000
Modified LLE embedding: Error = 0.003 | Score = 0.997
Hessian LLE embedding: Error = 0.829 | Score = 0.171
LTSA LLE embedding: cannot fit a decision tree. Nan Values found in projection.
MDS embedding: Error = 0.000 | Score = 1.000
Random Trees embedding: Error = 0.000 | Score = 1.000
```

```
Spectral embedding: Error = 0.000 | Score = 1.000
t-SNE embeedding: Error = 0.000 | Score = 1.000
NCA embedding: Error = 0.000 | Score = 1.000
```