math178_hw07_jkath

June 30, 2019

1 Manifold Learning Using Scikit Learn

Manifold learning is an approach to non-linear dimensionality reduction. Algorithms for this task are based on the idea that the dimensionality of many data sets is only artificially high. Various algorithms from the Scikit Manifold Learning module will be used to reduce higher dimensional non-linear data to lower dimensinality

NOTE: code examples are taken from the following sources

```
1) Python Data Science Handbook by Jake VanderPlas
2) Hands-on Machine Learning with Scikit-Learn and TensorFlow by A Geron
```

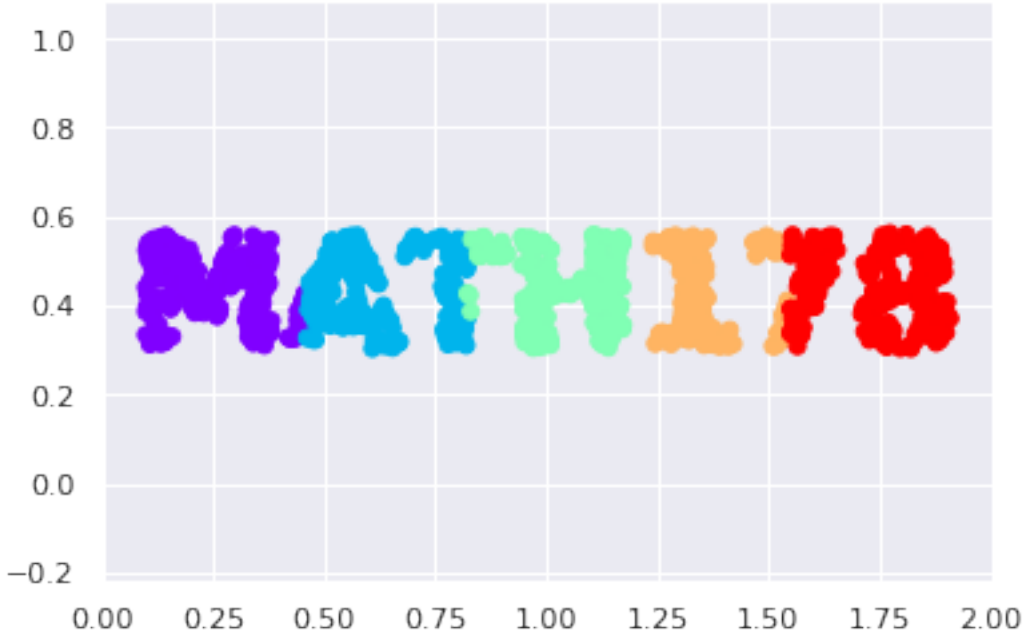
1.0.1 As a first example let's use the multidimensional scaling (MDS) and locally linear embedding (LLE) algorithms

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
```

[2]:
```
def make_text(N=1000, rseed=42):
    # Make a plot with "Math178" text; save as PNG
    fig, ax = plt.subplots(figsize=(8, 4))
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1)
    ax.axis('off')
    ax.text(0.5, 0.4, 'MATH178', va='center', ha='center', weight='bold',␣
 ↪size=100)
    fig.savefig('math178.png')
    plt.close(fig)
    # Open this PNG and draw random points from it
    from matplotlib.image import imread
    data = imread('math178.png')[::-1, :, 0].T
    rng = np.random.RandomState(rseed)
    X = rng.rand(4 * N, 2)
    i, j = (X * data.shape).astype(int).T
    mask = (data[i, j] < 1)
    X = X[mask]
    X[:, 0] *= (data.shape[0] / data.shape[1])
```

```
    X = X[:N]
    return X[np.argsort(X[:, 0])]
```

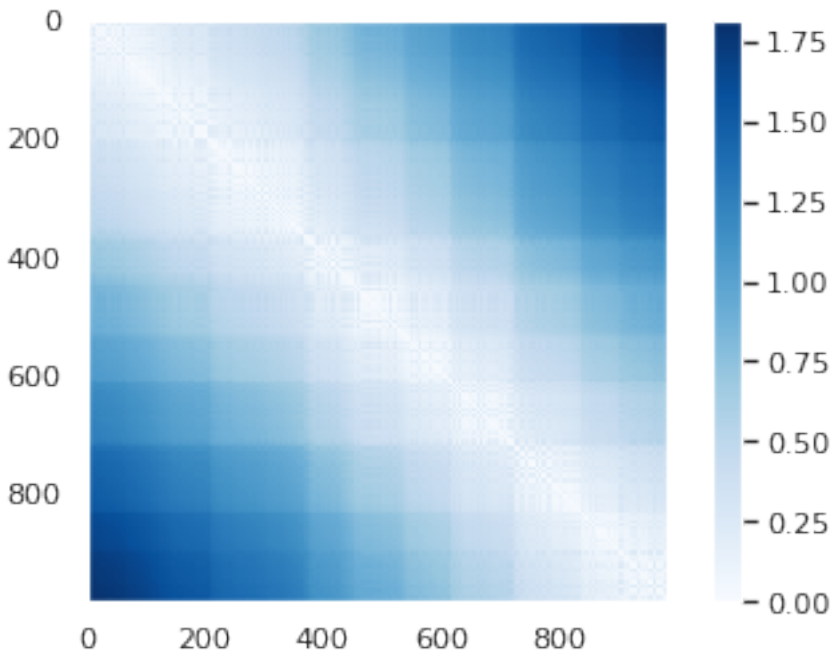
[3]:
```
X = make_text(2000)
colorize = dict(c=X[:, 0], cmap=plt.cm.get_cmap('rainbow', 5))
plt.scatter(X[:, 0], X[:, 1], **colorize)
plt.axis('equal');
```



[4]:
```
from sklearn.metrics import pairwise_distances
D = pairwise_distances(X)
D.shape
```

[4]:
```
(982, 982)
```

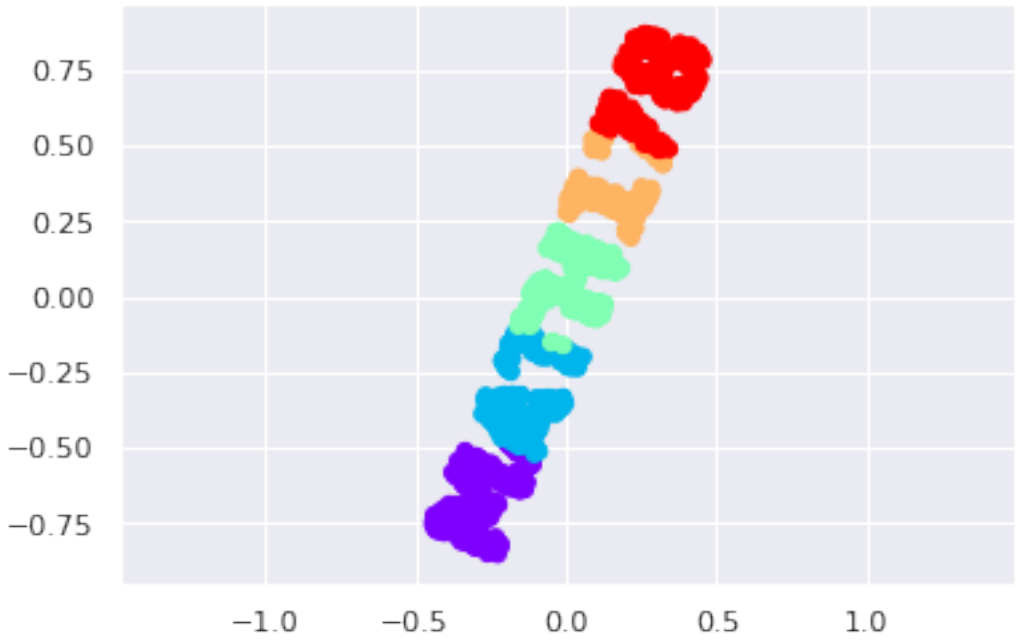
[5]:
```
plt.imshow(D, zorder=2, cmap='Blues', interpolation='nearest')
plt.colorbar();
```



Using a distance matrix to represent our data. This distance matrix gives us a representation of our data that is invariant to rotations and translations, but the visualization of the matrix above is not entirely intuitive. In the representation shown in this figure, we have lost any visible sign of the interesting structure in the data: the "Math178" that we saw before

1.0.2 MDS

[6]:
```
from sklearn.manifold import MDS
model = MDS(n_components=2, dissimilarity='precomputed', random_state=1)
out = model.fit_transform(D)
plt.scatter(out[:, 0], out[:, 1], **colorize)
plt.axis('equal');
```



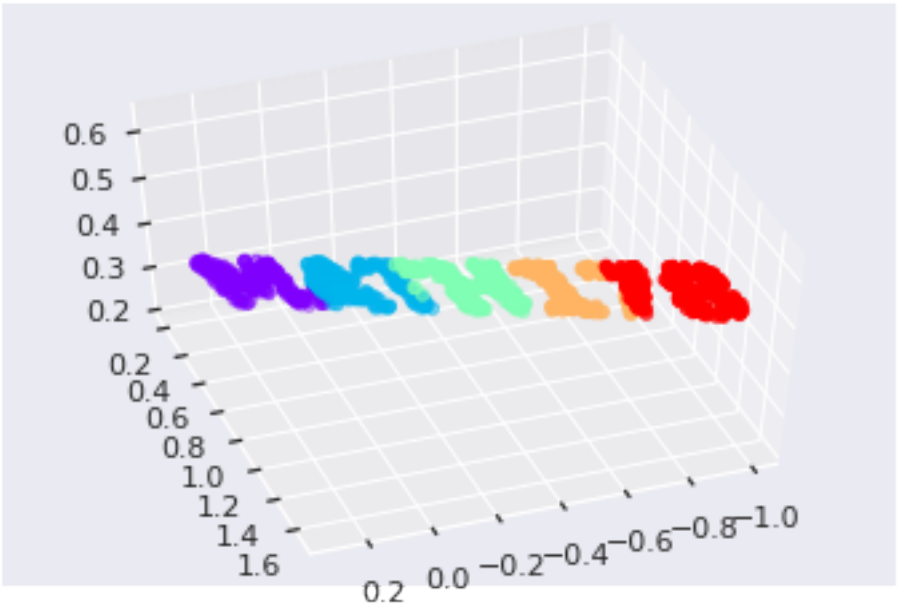
The MDS algorithm recovers one of the possible two-dimensional coordinate representations of our data, using *only* the $N \times N$ distance matrix describing the relationship between the data points

Now applying MDS again on a 3d projection of our data

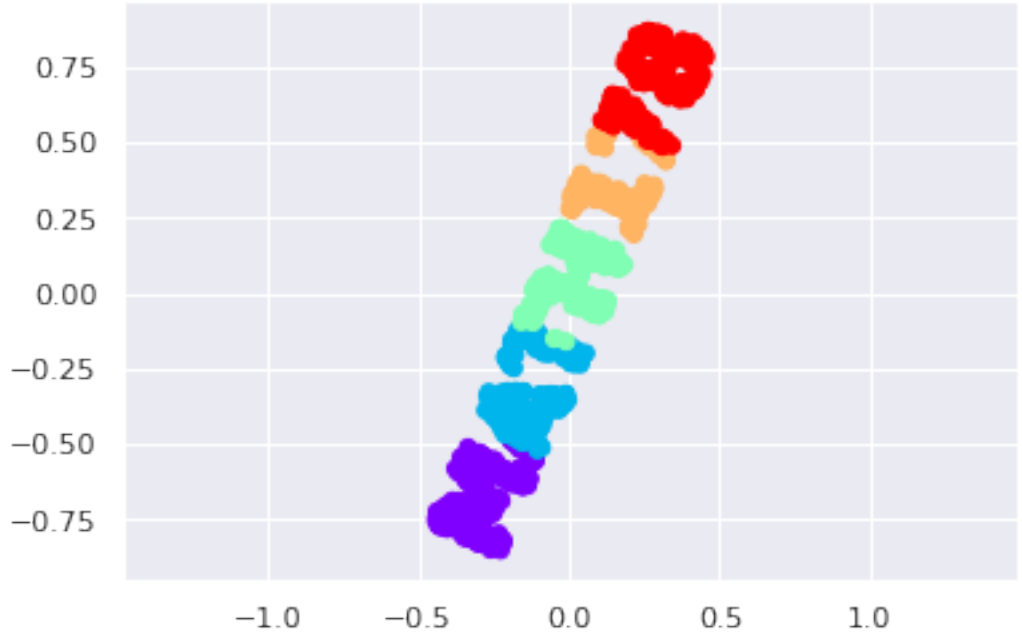
[7]:
```
def random_projection(X, dimension=3, rseed=42):
    assert dimension >= X.shape[1]
    rng = np.random.RandomState(rseed)
    C = rng.randn(dimension, dimension)
    e, V = np.linalg.eigh(np.dot(C, C.T))
    return np.dot(X, V[:X.shape[1]])

X3 = random_projection(X, 3)
X3.shape
```

[7]:
```
(982, 3)
```

[8]:
```
from mpl_toolkits import mplot3d
ax = plt.axes(projection='3d')
ax.scatter3D(X3[:, 0], X3[:, 1], X3[:, 2],
             **colorize)
ax.view_init(azim=70, elev=50)
```



[9]:
```
model = MDS(n_components=2, random_state=1)
out3 = model.fit_transform(X3)
plt.scatter(out3[:, 0], out3[:, 1], **colorize)
plt.axis('equal');
```

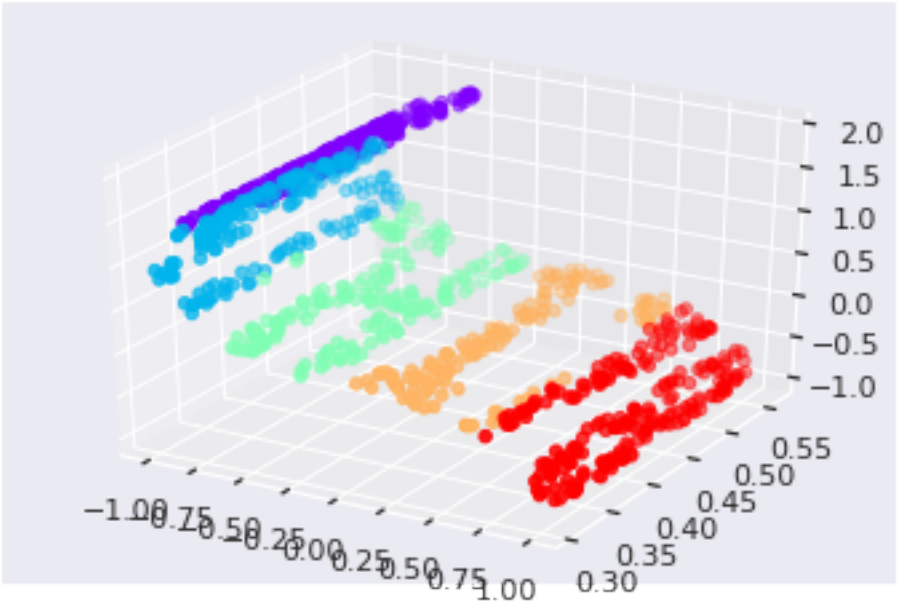


This time lets imbed our text data in a 3d S curve. We will use the LLE algorithm to recover the embedded two-dimensional manifold (as MDS in not well suited to this task)

[10]:
```
def make_text_s_curve(X):
    t = (X[:, 0] - 1.25) * .75 * np.pi
    x = np.sin(t)
    y = X[:, 1]
    z = np.sign(t) * (np.cos(t) - 1)
    return np.vstack((x, y, z)).T

XS = make_text_s_curve(X)
```

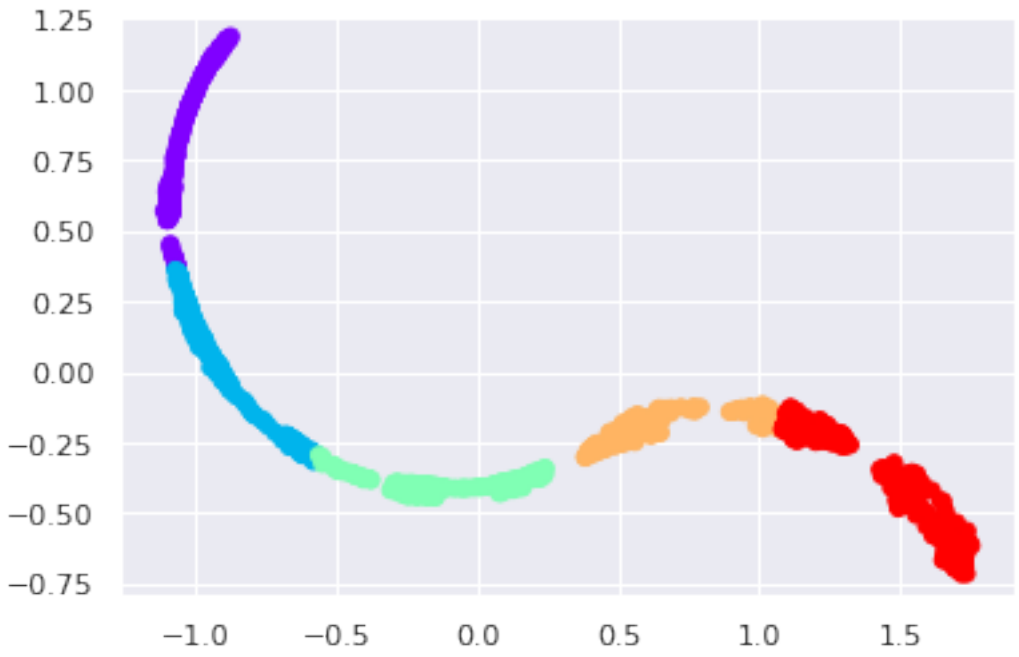
This is three-dimensional data, but we can see that the embedding is much more complicated

[11]:
```
from mpl_toolkits import mplot3d
ax = plt.axes(projection='3d')
ax.scatter3D(XS[:, 0], XS[:, 1], XS[:, 2], **colorize);
```



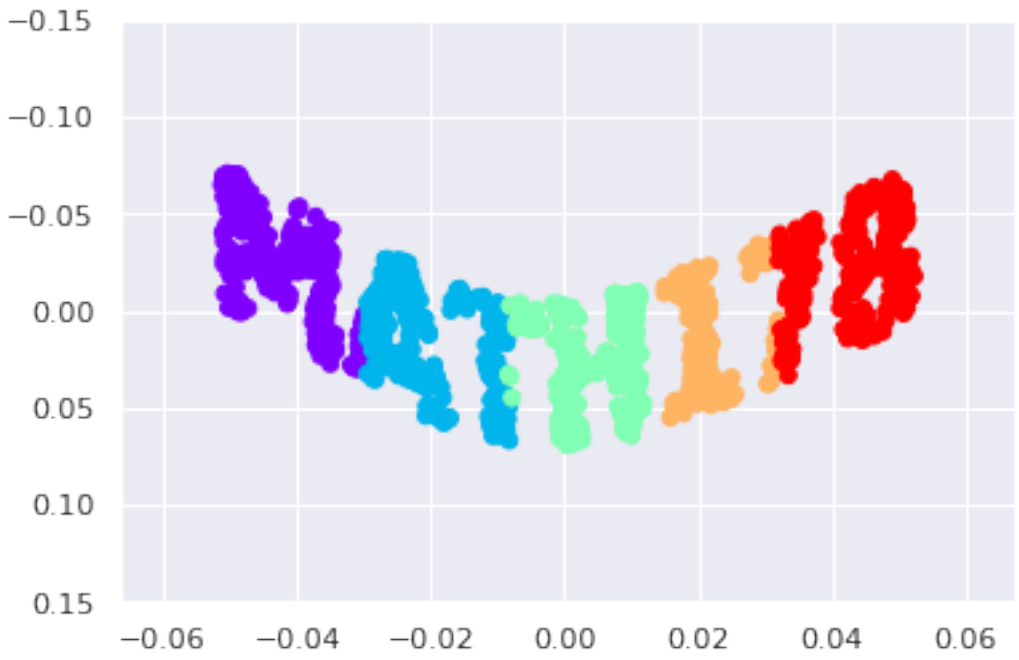
We can see MDS does not recover the imbedded data without losing the structure if the data

[12]:
```
from sklearn.manifold import MDS
model = MDS(n_components=2, random_state=2)
outS = model.fit_transform(XS)
plt.scatter(outS[:, 0], outS[:, 1], **colorize)
plt.axis('equal');
```



1.0.3 LLE

[13]:
```
from sklearn.manifold import LocallyLinearEmbedding
model = LocallyLinearEmbedding(n_neighbors=100, n_components=2,␣
 ↪method='modified',
                               eigen_solver='dense')
out = model.fit_transform(XS)

fig, ax = plt.subplots()
ax.scatter(out[:, 0], out[:, 1], **colorize)
ax.set_ylim(0.15, -0.15);
```



1.0.4 As a final example let's use the swill roll dataset with the Locally Linear Embedding (LLE), Isomap, t-Distributed Stochastic Neighbor Embedding (t-SNE) and Linear Discriminant Analysis (LDA) algoirthms

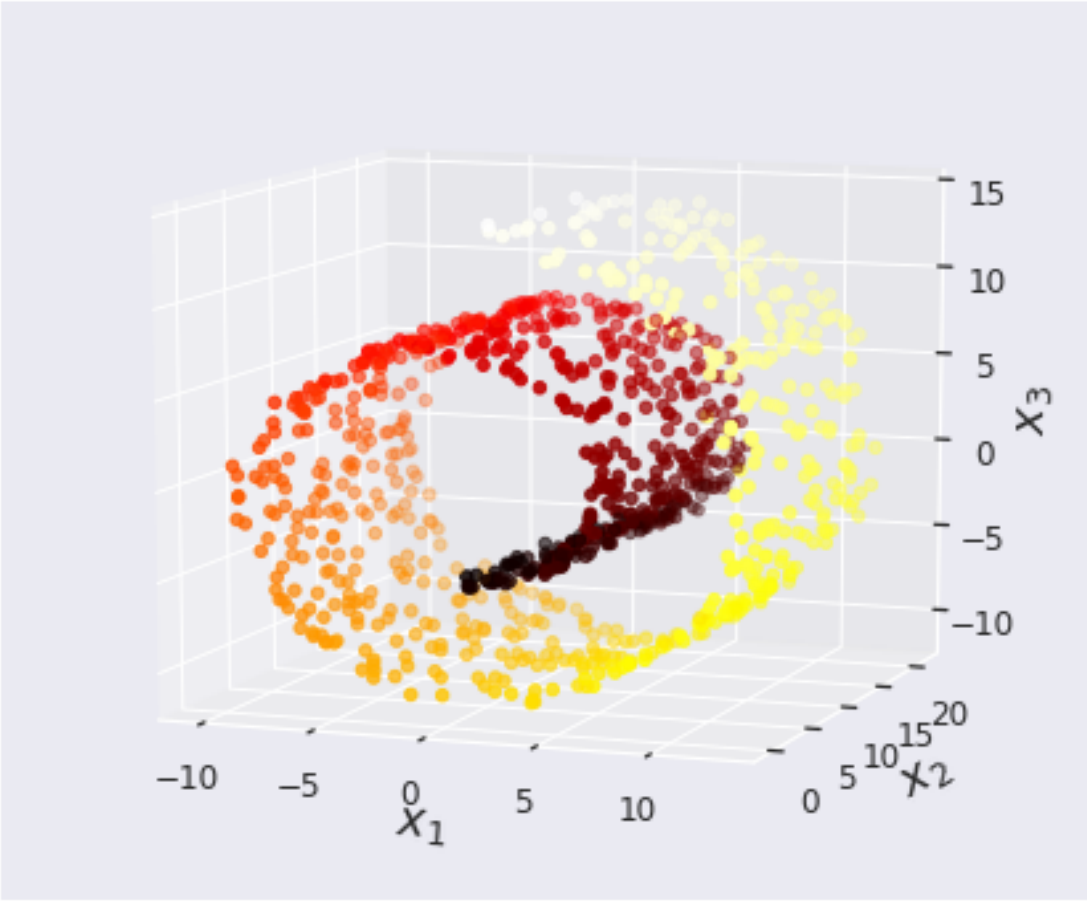
[14]:
```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = ""
```

```
def save_fig(fig_id, tight_layout=True):
    path = os.path.join(PROJECT_ROOT_DIR, "", CHAPTER_ID, fig_id + ".png")
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format='png', dpi=300)

# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

[15]:
```
from sklearn.datasets import make_swiss_roll
X, t = make_swiss_roll(n_samples=1000, noise=0.2, random_state=42)
```

[16]:
```
axes = [-11.5, 14, -2, 23, -12, 15]

fig = plt.figure(figsize=(6, 5))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=t, cmap=plt.cm.hot)
ax.view_init(10, -70)
ax.set_xlabel("$x_1$", fontsize=18)
ax.set_ylabel("$x_2$", fontsize=18)
ax.set_zlabel("$x_3$", fontsize=18)
ax.set_xlim(axes[0:2])
ax.set_ylim(axes[2:4])
ax.set_zlim(axes[4:6])

save_fig("swiss_roll_plot")
plt.show()
```

```
Saving figure swiss_roll_plot
```



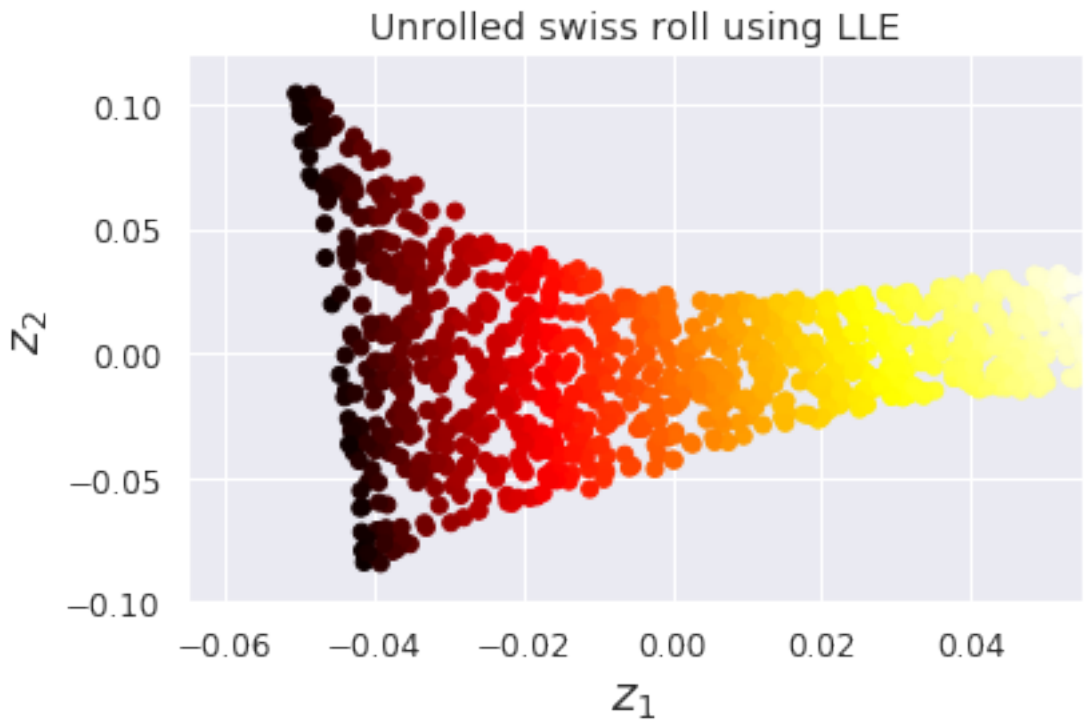
1.0.5 LLE

[17]:
```
X, t = make_swiss_roll(n_samples=1000, noise=0.2, random_state=41)
```

[18]:
```
from sklearn.manifold import LocallyLinearEmbedding

lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10, random_state=42)
X_reduced = lle.fit_transform(X)
```

[19]:
```
plt.title("Unrolled swiss roll using LLE", fontsize=14)
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=t, cmap=plt.cm.hot)
plt.xlabel("$z_1$", fontsize=18)
plt.ylabel("$z_2$", fontsize=18)
plt.axis([-0.065, 0.055, -0.1, 0.12])
plt.grid(True)

save_fig("lle_unrolling_plot")
plt.show()
```

```
Saving figure lle_unrolling_plot
```



The Swiss roll is completely unrolled and the distances between instances are locally well preserved. However, distances are not preserved on a larger scale: the left part of the unrolled Swiss roll is squeezed, while the right part is stretched. Nevertheless, LLE did a pretty good job at modeling the manifold

1.0.6 MDS, Isomap and t-SNE

Multidimensional Scaling (MDS) reduces dimensionality while trying to preserve the distances between the instances

Isomap creates a graph by connecting each instance to its nearest neighbors, then reduces dimensionality while trying to preserve the geodesic distances9 between the instances

t-Distributed Stochastic Neighbor Embedding (t-SNE) reduces dimensionality while trying to keep similar instances close and dissimilar instances apart. It is mostly used for visualization, in particular to visualize clusters of instances in high-dimensional space (e.g., to visualize the MNIST images in 2D)

Linear Discriminant Analysis (LDA) is actually a classification algorithm, but during training it learns the most discriminative axes between the classes, and these axes can then be used to define a hyperplane onto which to project the data. The benefit is that the projection will keep classes as far apart as possible, so LDA is a good technique to reduce dimensionality before running another classification algorithm such as an SVM classifier

[20]:
```
from sklearn.manifold import MDS
```

```
mds = MDS(n_components=2, random_state=42)
X_reduced_mds = mds.fit_transform(X)
```

[21]:
```
from sklearn.manifold import Isomap

isomap = Isomap(n_components=2)
X_reduced_isomap = isomap.fit_transform(X)
```

[22]:
```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=42)
X_reduced_tsne = tsne.fit_transform(X)
```

[23]:
```
from six.moves import urllib
try:
    from sklearn.datasets import fetch_openml
    mnist = fetch_openml('mnist_784', version=1)
    mnist.target = mnist.target.astype(np.int64)
except ImportError:
    from sklearn.datasets import fetch_mldata
    mnist = fetch_mldata('MNIST original')
```

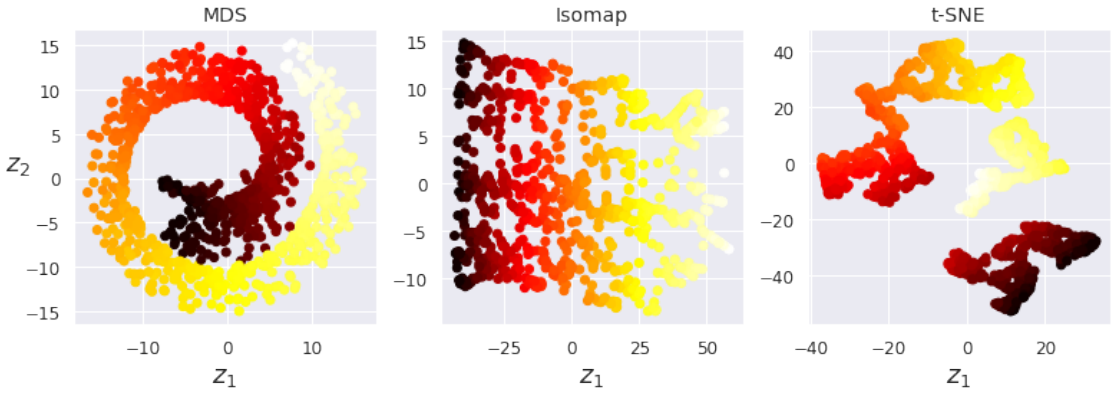
[24]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis(n_components=2)
X_mnist = mnist["data"]
y_mnist = mnist["target"]
lda.fit(X_mnist, y_mnist)
X_reduced_lda = lda.transform(X_mnist)
```

```
/home/jjkath/anaconda3/lib/python3.7/site-
packages/sklearn/discriminant_analysis.py:388: UserWarning: Variables are
collinear.
  warnings.warn("Variables are collinear.")
```

[25]:
```
titles = ["MDS", "Isomap", "t-SNE"]

plt.figure(figsize=(11,4))

for subplot, title, X_reduced in zip((131, 132, 133), titles,
                                     (X_reduced_mds, X_reduced_isomap,␣
 ↪X_reduced_tsne)):
    plt.subplot(subplot)
    plt.title(title, fontsize=14)
    plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=t, cmap=plt.cm.hot)
    plt.xlabel("$z_1$", fontsize=18)
    if subplot == 131:
        plt.ylabel("$z_2$", fontsize=18, rotation=0)
    plt.grid(True)
```

```
save_fig("other_dim_reduction_plot")
plt.show()
```

```
Saving figure other_dim_reduction_plot
```



[ ]: