

**School of Design and Informatics**

**Applied Mathematics and Artificial Intelligence**

**(MAT501)**

**Math in AI – The Singular Value Decomposition and Principal Component Analysis**

**(Practical for Week 2)**

Session Outline

* 1. Introduction
  2. Principal Component Analysis and Singular Value Decomposition of a mathematical object
  3. Comparison of PCA with other machine learning methods

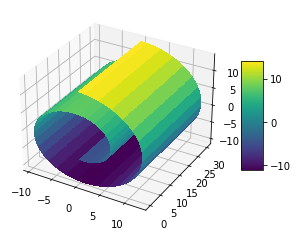
Dr. Kean Lee Kang, School of Design and Informatics

**Practical for Week 2: The Singular Value Decomposition and Principal Component Analysis**

**Introduction**

For this practical, we will use a mathematical object known as a “swiss roll” as our example dataset.

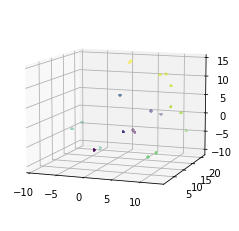
The swiss roll can be visualized as follows:



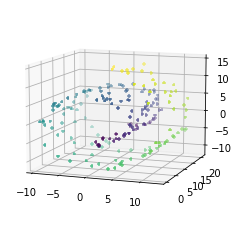
The machine learning algorithms do not “know” about the swiss roll. All they see are a collection of data points (samples) taken from the surface of the swiss roll.

If we randomly sample N points on the surface of the swiss roll, we get the following results:

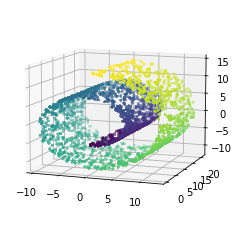
N=20



N=200

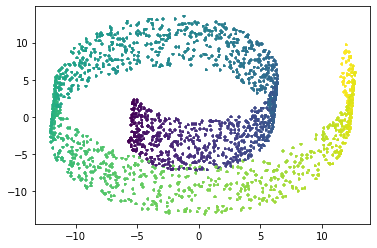


N=2000



As we can see, the more points we sample, the more the cloud of points looks like a swiss roll. Generally speaking, in AI and machine learning, the more data we have, the better the AI can understand the underlying phenomena that we are seeking to model.

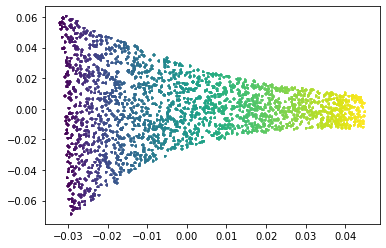
One key attribute of AI is the ability to simplify or extract the most important elements in a dataset. We learnt in the lectures that the singular value decomposition (SVD) is one way to do so, and the SVD is a solution to the principal component analysis (PCA) problem. If we run PCA on the sampled data points from the swiss roll, we obtain the following from the first two modes of the PCA:



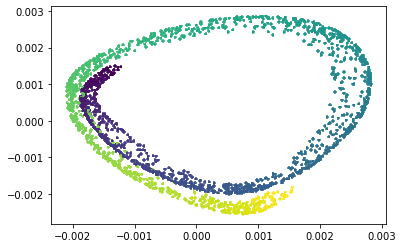
This can be thought of as a projection of the 3D swiss roll onto a 2D plane. Clearly, the PCA has successfully captured the spiral structure of the swiss roll.

We can compare the PCA against several other “manifold learning methods”, such as locally linear embedding and spectral embedding below.

Locally linear embedding:



Spectral embedding:



**Exercises**

1. Copy and paste the code in Appendix 1 into your Python IDE and run it. (Alternatively, download and run the Python script from MyLearningSpace: “MAT501 Practical for Week 3 SVD.py”.) Check that you obtain figures similar to those in the introduction. Note that the figures that appear on your machine may not be exactly the same as those in this guide because of random sampling.
2. Compare the results of PCA with those of locally linear embedding and spectral embedding. What are the strengths and weaknesses of each method in terms of their summarization of the key attributes of the swiss roll?
3. In the code, the number of random samples is N=2000. Try to increase or decrease N and rerun the code. How are the results affected?
4. (Optional, Advanced) So far, we have used the Scikit-learn library to compute our PCA. It is possible to compute the PCA directly using the SVD of the data matrix. Start by appending the code below to the end of your Python script. What is this code doing? How might you use the matrices U, S, V and/or Sigma to find the PCA?

U,S,V = np.linalg.svd(X)

m, n = np.shape(X)

# Create full Sigma matrix from singular values in S

sigma = np.zeros((m, n))

for i in range(min(m, n)):

sigma[i, i] = S[i]

X1 = np.dot(U, np.dot(sigma, V))

assert(np.allclose(X, X1)) #check that the SVD matrices are correct by recovering X

1. (Optional, Advanced) We can also attempt to perform clustering on the swiss roll dataset. To begin with, run the Python script in Appendix 2, “MAT501 Practical for Clustering.py”. Try changing the number of clusters, n\_clusters being found by the K-Means algorithm. Is the cluster assignment satisfactory? If it is not, try applying locally linear embedding first before clustering the results. Is the new cluster assignment better?

**Appendix 1: MAT501 Practical for SVD.py**

# -\*- coding: utf-8 -\*-

"""

MAT501 Practical 2

The Singular Value Decomposition and Principal Component Analysis

Kean Lee Kang

Abertay University

2021

"""

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

import math

from sklearn.manifold import LocallyLinearEmbedding

from sklearn.manifold import SpectralEmbedding

from sklearn.decomposition import PCA

N = 2000;

K = 12;

d = 2;

# PLOT TRUE MANIFOLD

tt0 = (3\*math.pi/2)\*(1+2\*np.linspace(0,1,num=501))

hh = np.linspace(0,1,num=99)\*30

xx = np.outer(np.multiply(tt0,np.cos(tt0)),np.ones(np.shape(hh)));

yy = np.outer(np.ones(np.shape(tt0)),hh)

zz = np.outer(np.multiply(tt0,np.sin(tt0)),np.ones(np.shape(hh)));

cc = np.outer(tt0,np.ones(np.shape(hh)))

# Plot the surface.

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

surf = ax.plot\_surface(xx, yy, zz, cmap=cm.viridis, #perceptually uniform cmap

linewidth=0, antialiased=False)

# Add a color bar which maps values to colors.

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()

# GENERATE SAMPLED DATA

tt = (3\*math.pi/2)\*(1+2\*np.random.random\_sample(N))

height = 21\*np.random.random\_sample(N)

X = np.column\_stack((np.multiply(tt,np.cos(tt)),height,np.multiply(tt,np.sin(tt))))

# SCATTERPLOT OF SAMPLED DATA

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(X[:,0],X[:,1],X[:,2],s=12,c=tt,marker='+')

ax.view\_init(elev=10, azim=290)

#for ii in range(0,360,1):

# ax.view\_init(elev=10., azim=ii)

# plt.savefig('movie%d.png' % ii)

plt.show()

# RUN LLE ALGORITHM

embedding = LocallyLinearEmbedding(n\_neighbors=12,n\_components=2)

Y = embedding.fit\_transform(X)

# SCATTERPLOT OF EMBEDDING

fig = plt.figure()

plt.scatter(Y[:,0],Y[:,1],s=12,c=tt,marker='+')

plt.show()

# RUN SPECTRAL EMBEDDING ALGORITHM

spectral = SpectralEmbedding(n\_components=2)

Y\_SPEC = spectral.fit\_transform(X)

# SCATTERPLOT OF SPECTRAL EMBEDDING

fig = plt.figure()

plt.scatter(Y\_SPEC[:,0],Y\_SPEC[:,1],s=12,c=tt,marker='+')

plt.show()

# PCA via SCIKIT LEARN FUNCTION

pca = PCA()

Y\_PCA = pca.fit\_transform(X)

# SCATTERPLOT OF PCA with d=2

fig = plt.figure()

plt.scatter(Y\_PCA[:,0],Y\_PCA[:,1],s=12,c=tt,marker='+')

plt.show()

**Appendix 2: MAT501 Practical for Clustering.py**

# -\*- coding: utf-8 -\*-

"""

MAT501 Practical 2

Clustering

Kean Lee Kang

Abertay University

2021

"""

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

import math

from sklearn.manifold import LocallyLinearEmbedding

from sklearn.manifold import SpectralEmbedding

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

N = 2000

K = 12

d = 2

n\_clusters = 3

random\_state = 41

# PLOT TRUE MANIFOLD

tt0 = (3\*math.pi/2)\*(1+2\*np.linspace(0,1,num=501))

hh = np.linspace(0,1,num=99)\*30

xx = np.outer(np.multiply(tt0,np.cos(tt0)),np.ones(np.shape(hh)));

yy = np.outer(np.ones(np.shape(tt0)),hh)

zz = np.outer(np.multiply(tt0,np.sin(tt0)),np.ones(np.shape(hh)));

cc = np.outer(tt0,np.ones(np.shape(hh)))

# Plot the surface.

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

surf = ax.plot\_surface(xx, yy, zz, cmap=cm.viridis, #perceptually uniform cmap

linewidth=0, antialiased=False)

# Add a color bar which maps values to colors.

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()

# GENERATE SAMPLED DATA

tt = (3\*math.pi/2)\*(1+2\*np.random.random\_sample(N))

height = 21\*np.random.random\_sample(N)

X = np.column\_stack((np.multiply(tt,np.cos(tt)),height,np.multiply(tt,np.sin(tt))))

# SCATTERPLOT OF SAMPLED DATA

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(X[:,0],X[:,1],X[:,2],s=12,c=tt,marker='+')

ax.view\_init(elev=10, azim=290)

#for ii in range(0,360,1):

# ax.view\_init(elev=10., azim=ii)

# plt.savefig('movie%d.png' % ii)

plt.show()

# CLUSTERING OF SAMPLED DATA

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=random\_state)

y\_pred = kmeans.fit\_predict(X)

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(X[:,0],X[:,1],X[:,2],s=12,c=y\_pred,marker='+')

centroids = kmeans.cluster\_centers\_

ax.scatter(centroids[:,0],centroids[:,1],centroids[:,2], marker="x", s=169, color="k")

ax.view\_init(elev=10, azim=290)

plt.title("KMeans with n\_clusters="+str(len(centroids)))

plt.show()