实验四 数据降维与可视化

一、实验目的及要求

1、掌握 PCA、LPP、MDS、ISOMAP，LLE，等数据降维与可视化的方法。

2、掌握 PCA 与 LPP 数据降维算法的工作原理与实现方法。

3、掌握 MDS 以及 ISOMAP 数据可视化的方法，理解测地线距离的计算方法。

4、理解 LLE 算法的基本原理，理解流形学习的一般思路

二、预习要求

阅读本实验例程部分，实现基本的逻辑回归以及 BP 神经网络分类方法，以便够

充分利用实验时间编程调试。

三、实验设备

硬件：PC机。

软件：Python 及相关集成开发环境。

四、实验内容

利用利用 Python 编程实现基于决策树分类算法，并将训练得到决策树绘制来。

具体要求如下：

1. 编程实现 PCA 算法对鸢尾花数据集进行降维与可视化
2. 实验报告内容要求
3. 1. 列出编写的 python 代码。对主要的语句进行注释

PCA

import numpy as np

import matplotlib.pyplot as plt

*# data 输入数据 维度 [N，D]*

*# n\_dim: 降维后的维度*

*# 返回 [N,n\_dim]*

def pca(data, n\_dim):

N,D = np.shape(data)

data = data - np.mean(data, axis = 0, keepdims = True)

C = np.dot(data.T, data)/(N-1) *# [D,D]*

*# 计算特征值和特征向量*

eig\_values, eig\_vector = np.linalg.eig(C)

*# 将特征值进行排序选取 n\_dim 个较大的特征值*

indexs\_ = np.argsort(-eig\_values)[:n\_dim]

*# 选取相应的特征向量组成降维矩阵*

picked\_eig\_vector = eig\_vector[:, indexs\_] *# [D,n\_dim]*

*# 对数据进行降维*

data\_ndim = np.dot(data, picked\_eig\_vector)

return data\_ndim, picked\_eig\_vector

def draw\_pic(datas,labs):

plt.cla()

unque\_labs = np.unique(labs)

colors = [plt.cm.Spectral(each)

for each in np.linspace(0, 1,len(unque\_labs))]

p=[]

legends = []

for i in range(len(unque\_labs)):

index = np.where(labs==unque\_labs[i])

pi = plt.scatter(datas[index, 0], datas[index, 1], c =[colors[i]] )

p.append(pi)

legends.append(unque\_labs[i])

plt.legend(p, legends)

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

*# 加载数据*

data = np.loadtxt("iris.data",dtype="str",delimiter=',')

feas = data[:,:-1]

feas = np.float32(feas)

labs = data[:,-1]

*# 进行降维*

data\_2d, picked\_eig\_vector= pca(feas, 2)

*#绘图*

draw\_pic(data\_2d,labs)

LPP

*# coding:utf-8*

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.datasets import load\_digits, load\_iris

from sklearn.datasets import make\_swiss\_roll

*# x 维度 [N,D]*

def cal\_pairwise\_dist(X):

N,D = np.shape(X)

tile\_xi = np.tile(np.expand\_dims(X,1),[1,N,1])

tile\_xj = np.tile(np.expand\_dims(X,axis=0),[N,1,1])

dist = np.sum((tile\_xi-tile\_xj)\*\*2,axis=-1)

*#返回任意两个点之间距离*

return dist

def rbf(dist, t = 1.0):

*'''*

*rbf kernel function*

*'''*

return np.exp(-(dist/t))

def cal\_rbf\_dist(data, n\_neighbors = 10, t = 1):

dist = cal\_pairwise\_dist(data)

dist[dist < 0] = 0

N = dist.shape[0]

rbf\_dist = rbf(dist, t)

W = np.zeros([N, N])

for i in range(N):

index\_ = np.argsort(dist[i])[1:1 + n\_neighbors]

W[i, index\_] = rbf\_dist[i, index\_]

W[index\_, i] = rbf\_dist[index\_, i]

return W

*# X 输入高维数据 格式 [N，D]*

*# n\_neighbors K近邻的数目*

*# t 权重计算的参数*

def lpp(X,n\_dims = 2,n\_neighbors = 30, t = 1.0):

N = X.shape[0]

W = cal\_rbf\_dist(X, n\_neighbors, t)

D = np.zeros\_like(W)

for i in range(N):

D[i,i] = np.sum(W[i])

L = D - W

XDXT = np.dot(np.dot(X.T, D), X)

XLXT = np.dot(np.dot(X.T, L), X)

eig\_val, eig\_vec = np.linalg.eig(np.dot(np.linalg.pinv(XDXT), XLXT))

sort\_index\_ = np.argsort(np.abs(eig\_val))

eig\_val = eig\_val[sort\_index\_]

print("eig\_val[:10]", eig\_val[:10])

j = 0

while eig\_val[j] < 1e-6:

j+=1

print("j: ", j)

sort\_index\_ = sort\_index\_[j:j+n\_dims]

eig\_val\_picked = eig\_val[j:j+n\_dims]

print(eig\_val\_picked)

A = eig\_vec[:, sort\_index\_]

Y = np.dot(X, A)

return Y

def scatter\_3d(X, y):

fig = plt.figure()

ax = plt.axes(projection='3d')

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, 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)

plt.show(block=False)

if \_\_name\_\_ == '\_\_main\_\_':

*# #1 测试瑞士卷数据*

*# X, Y = make\_swiss\_roll(n\_samples=1000)*

*# scatter\_3d(X, Y)*

*# n\_neighbors = 10*

*# 2 测试 load\_digits 数据*

*# X = load\_digits().data*

*# Y = load\_digits().target*

*# n\_neighbors = 5*

*#3 测试 load\_iris 数据*

X = load\_iris().data

Y = load\_iris().target

n\_neighbors = 5

dist = cal\_pairwise\_dist(X)

max\_dist = np.max(dist)

data\_2d\_LPP = lpp(X, n\_neighbors = n\_neighbors, t = 0.01\*max\_dist)

data\_2d\_PCA = PCA(n\_components=2).fit\_transform(X)

plt.figure(figsize=(12,6))

plt.subplot(121)

plt.title("LPP")

plt.scatter(data\_2d\_LPP[:, 0], data\_2d\_LPP[:, 1], c = Y)

plt.subplot(122)

plt.title("PCA")

plt.scatter(data\_2d\_PCA[:, 0], data\_2d\_PCA[:, 1], c = Y)

plt.show()

MDS

import numpy as np

import matplotlib.pyplot as plt

*# x 维度 [N,D]*

def cal\_pairwise\_dist(x):

N,D = np.shape(x)

dist = np.zeros([N,N])

for i in range(N):

for j in range(N):

*#dist[i,j] = np.dot((x[i]-x[j]),(x[i]-x[j]).T)*

*#dist[i,j] = np.sqrt(np.dot((x[i]-x[j]),(x[i]-x[j]).T))*

dist[i,j] = np.sum(np.abs(x[i]-x[j]))

*#返回任意两个点之间距离*

return dist

*# dist N\*N 距离矩阵样本点两两之间的距离*

*# n\_dims 降维*

*# 返回 降维后的数据*

def my\_mds(dist, n\_dims):

n,n = np.shape(dist)

dist[dist < 0 ] = 0

dist = dist\*\*2

T1 = np.ones((n,n))\*np.sum(dist)/n\*\*2

T2 = np.sum(dist, axis = 1, keepdims=True)/n

T3 = np.sum(dist, axis = 0, keepdims=True)/n

B = -(T1 - T2 - T3 + dist)/2

eig\_val, eig\_vector = np.linalg.eig(B)

index\_ = np.argsort(-eig\_val)[:n\_dims]

picked\_eig\_val = eig\_val[index\_].real

picked\_eig\_vector = eig\_vector[:, index\_]

return picked\_eig\_vector\*picked\_eig\_val\*\*(0.5)

def draw\_pic(datas,labs):

plt.cla()

unque\_labs = np.unique(labs)

colors = [plt.cm.Spectral(each)

for each in np.linspace(0, 1,len(unque\_labs))]

p=[]

legends = []

for i in range(len(unque\_labs)):

index = np.where(labs==unque\_labs[i])

pi = plt.scatter(datas[index, 0], datas[index, 1], c =[colors[i]] )

p.append(pi)

legends.append(unque\_labs[i])

plt.legend(p, legends)

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

*# 加载数据*

data = np.loadtxt("iris.data",dtype="str",delimiter=',')

feas = data[:,:-1]

feas = np.float32(feas)

labs = data[:,-1]

*# 计算距离*

dist = cal\_pairwise\_dist(feas)

*# 进行降维*

data\_2d = my\_mds(dist, 2)

*#绘图*

draw\_pic(data\_2d,labs)

ISOMAP

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_s\_curve

from sklearn.manifold import Isomap

from tqdm import tqdm

from mpl\_toolkits.mplot3d import Axes3D

*# x 维度 [N,D]*

def cal\_pairwise\_dist(x):

N,D = np.shape(x)

dist = np.zeros([N,N])

for i in range(N):

for j in range(N):

*# dist[i,j] = np.dot((x[i]-x[j]),(x[i]-x[j]).T)*

dist[i,j] = np.sqrt(np.dot((x[i]-x[j]),(x[i]-x[j]).T))

*# dist[i,j] = np.sum(np.abs(x[i]-x[j]))*

*#返回任意两个点之间距离*

return dist

*# 构建最短路径图*

def floyd(D,n\_neighbors=15):

Max = np.max(D)\*1000

n1,n2 = D.shape

k = n\_neighbors

D1 = np.ones((n1,n1))\*Max

D\_arg = np.argsort(D,axis=1)

for i in range(n1):

D1[i,D\_arg[i,0:k+1]] = D[i,D\_arg[i,0:k+1]]

for k in tqdm(range(n1)):

for i in range(n1):

for j in range(n1):

if D1[i,k]+D1[k,j]<D1[i,j]:

D1[i,j] = D1[i,k]+D1[k,j]

return D1

def my\_mds(dist, n\_dims):

*# dist (n\_samples, n\_samples)*

dist = dist\*\*2

n = dist.shape[0]

T1 = np.ones((n,n))\*np.sum(dist)/n\*\*2

T2 = np.sum(dist, axis = 1)/n

T3 = np.sum(dist, axis = 0)/n

B = -(T1 - T2 - T3 + dist)/2

eig\_val, eig\_vector = np.linalg.eig(B)

index\_ = np.argsort(-eig\_val)[:n\_dims]

picked\_eig\_val = eig\_val[index\_].real

picked\_eig\_vector = eig\_vector[:, index\_]

return picked\_eig\_vector\*picked\_eig\_val\*\*(0.5)

*# dist N\*N 距离矩阵样本点两两之间的距离*

*# n\_dims 降维*

*# 返回 降维后的数据*

*# def my\_mds(dist, n\_dims):*

*# n,n = np.shape(dist)*

*# dist[dist < 0 ] = 0*

*# T1 = np.ones((n,n))\*np.sum(dist)/n\*\*2*

*# T2 = np.sum(dist, axis = 1, keepdims=True)/n*

*# T3 = np.sum(dist, axis = 0, keepdims=True)/n*

*# B = -(T1 - T2 - T3 + dist)/2*

*# eig\_val, eig\_vector = np.linalg.eig(B)*

*# index\_ = np.argsort(-eig\_val)[:n\_dims]*

*# picked\_eig\_val = eig\_val[index\_].real*

*# picked\_eig\_vector = eig\_vector[:, index\_]*

*# return picked\_eig\_vector\*picked\_eig\_val\*\*(0.5)*

def my\_Isomap(D,n=2,n\_neighbors=30):

D\_floyd=floyd(D, n\_neighbors)

data\_n = my\_mds(D\_floyd, n\_dims=n)

return data\_n

def scatter\_3d(X, y):

fig = plt.figure()

ax = plt.axes(projection='3d')

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, 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)

plt.show(block=False)

if \_\_name\_\_ == "\_\_main\_\_":

X, Y = make\_s\_curve(n\_samples = 500,

noise = 0.1,

random\_state = 42)

scatter\_3d(X,Y)

*# 计算距离*

dist = cal\_pairwise\_dist(X)

*# MDS 降维*

data\_MDS = my\_mds(dist, 2)

plt.figure()

plt.title("my\_MSD")

plt.scatter(data\_MDS[:, 0], data\_MDS[:, 1], c = Y)

plt.show(block=False)

*# ISOMAP 降维*

data\_ISOMAP = my\_Isomap(dist, 2, 10)

plt.figure()

plt.title("my\_Isomap")

plt.scatter(data\_ISOMAP[:, 0], data\_ISOMAP[:, 1], c = Y)

plt.show(block=False)

data\_ISOMAP2 = Isomap(n\_neighbors = 10, n\_components = 2).fit\_transform(X)

plt.figure()

plt.title("sk\_Isomap")

plt.scatter(data\_ISOMAP2[:, 0], data\_ISOMAP2[:, 1], c = Y)

plt.show(block=False)

plt.show()

*# # 加载数据*

*# data = np.loadtxt("iris.data",dtype="str",delimiter=',')*

*# feas = data[:,:-1]*

*# feas = np.float32(feas)*

*# labs = data[:,-1]*

*# # 计算距离*

*# dist = cal\_pairwise\_dist(feas)*

*# # 进行降维*

*# data\_2d = my\_mds(dist, 2)*

*# #绘图*

*# draw\_pic(data\_2d,labs)*

*LLE*

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_swiss\_roll

from tqdm import tqdm

from sklearn.manifold import LocallyLinearEmbedding

from mpl\_toolkits.mplot3d import Axes3D

*# x 维度 [N,D]*

def cal\_pairwise\_dist(x):

N,D = np.shape(x)

dist = np.zeros([N,N])

for i in range(N):

for j in range(N):

dist[i,j] = np.sqrt(np.dot((x[i]-x[j]),(x[i]-x[j]).T))

*#返回任意两个点之间距离*

return dist

*# 获取每个样本点的 n\_neighbors个临近点的位置*

def get\_n\_neighbors(data, n\_neighbors = 10):

dist = cal\_pairwise\_dist(data)

dist[dist < 0] = 0

N = dist.shape[0]

Index = np.argsort(dist,axis=1)[:,1:n\_neighbors+1]

return Index

*# data : N,D*

def lle(data, n\_dims = 2, n\_neighbors = 10):

N,D = np.shape(data)

if n\_neighbors > D:

tol = 1e-3

else:

tol = 0

*# 获取 n\_neighbors个临界点的位置*

Index\_NN = get\_n\_neighbors(data,n\_neighbors)

*# 计算重构权重*

w = np.zeros([N,n\_neighbors])

for i in range(N):

X\_k = data[Index\_NN[i]] *#[k,D]*

X\_i = [data[i]] *#[1,D]*

I = np.ones([n\_neighbors,1])

Si = np.dot((np.dot(I,X\_i)-X\_k), (np.dot(I,X\_i)-X\_k).T)

*# 为防止对角线元素过小*

Si = Si+np.eye(n\_neighbors)\*tol\*np.trace(Si)

Si\_inv = np.linalg.pinv(Si)

w[i] = np.dot(I.T,Si\_inv)/(np.dot(np.dot(I.T,Si\_inv),I))

*# 计算 W*

W = np.zeros([N,N])

for i in range(N):

W[i,Index\_NN[i]] = w[i]

I\_N = np.eye(N)

C = np.dot((I\_N-W).T,(I\_N-W))

*# 进行特征值的分解*

eig\_val, eig\_vector = np.linalg.eig(C)

index\_ = np.argsort(eig\_val)[1:n\_dims+1]

y = eig\_vector[:,index\_]

return y

def scatter\_3d(X, y):

fig = plt.figure()

ax = plt.axes(projection='3d')

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, 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)

plt.show(block=False)

if \_\_name\_\_ == "\_\_main\_\_":

X, Y = make\_swiss\_roll(n\_samples=500)

scatter\_3d(X,Y)

data\_2d = lle(X, n\_dims = 2, n\_neighbors = 12)

print(data\_2d.shape)

plt.figure()

plt.title("my\_LLE")

plt.scatter(data\_2d[:, 0], data\_2d[:, 1], c = Y,cmap=plt.cm.hot)

plt.show(block=False)

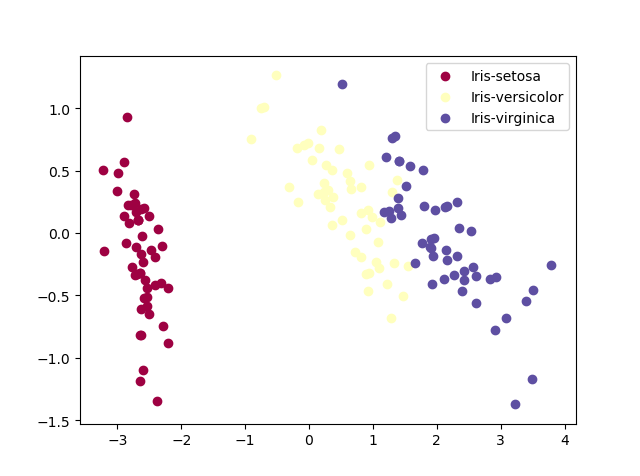
data\_2d\_sk = LocallyLinearEmbedding(n\_components=2, n\_neighbors = 12).fit\_transform(X)

plt.figure()

plt.title("my\_LLE\_sk")

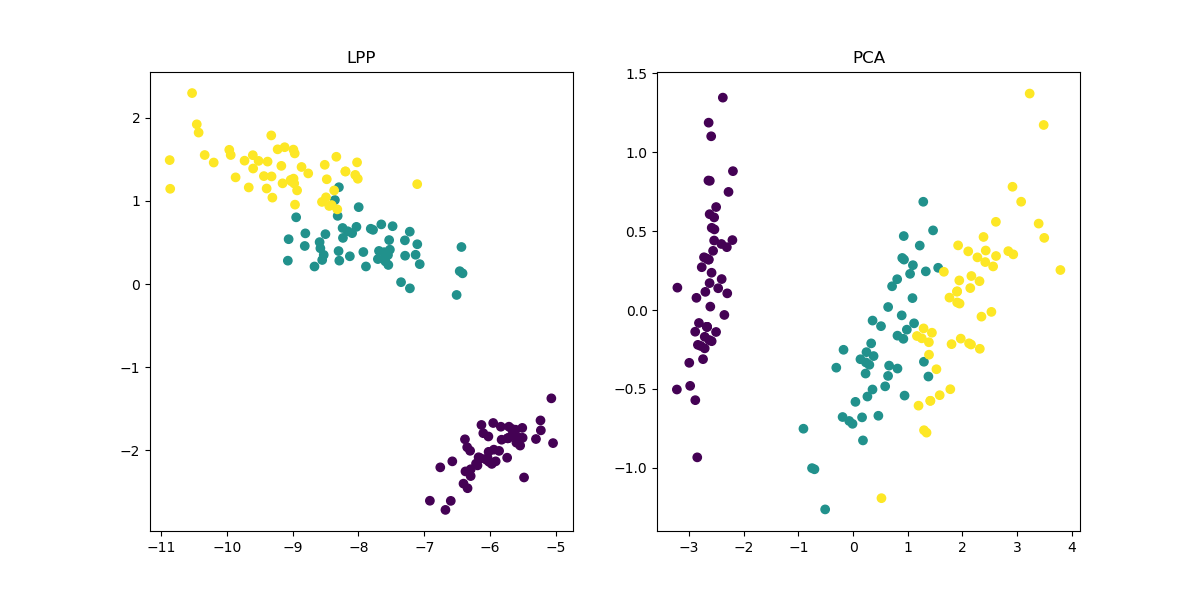
plt.scatter(data\_2d[:, 0], data\_2d[:, 1], c = Y,cmap=plt.cm.hot)

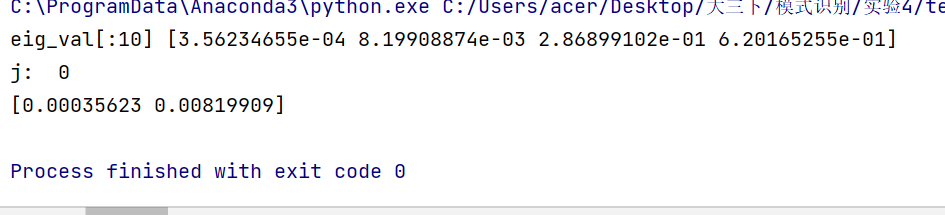
plt.show()



（2）编程实现 LPP 算法，利用该算法鸢尾花数据集进行降维与可视化，并与 PCA 的

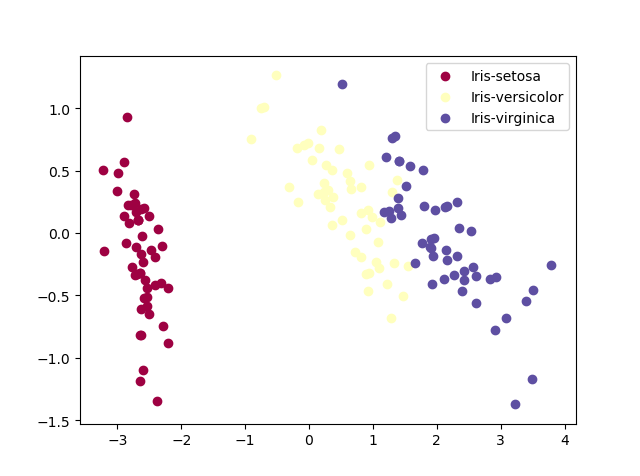
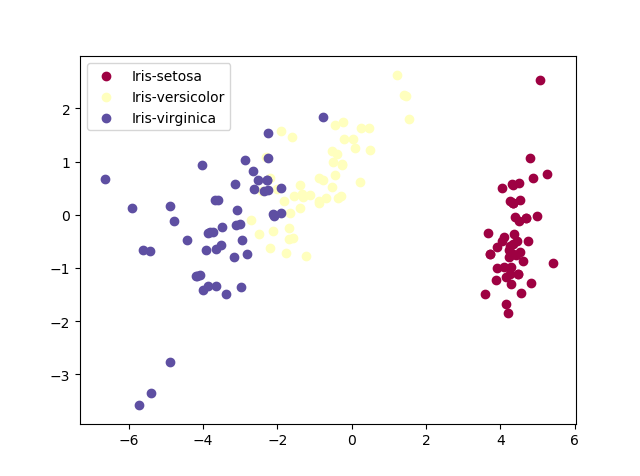
降维效果进行比较



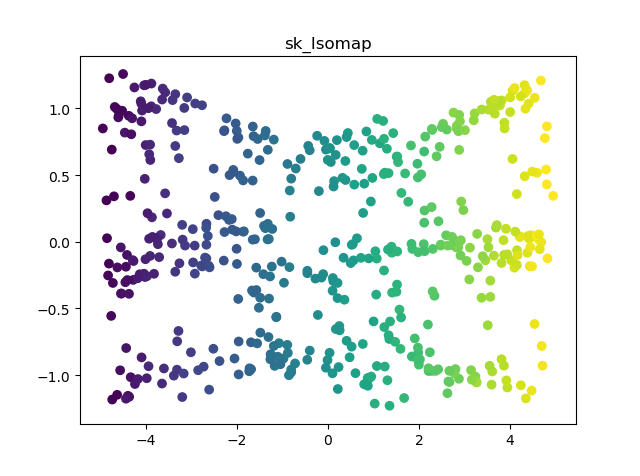
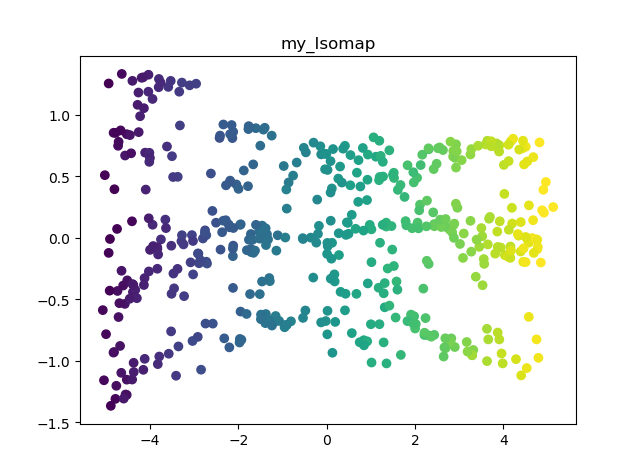
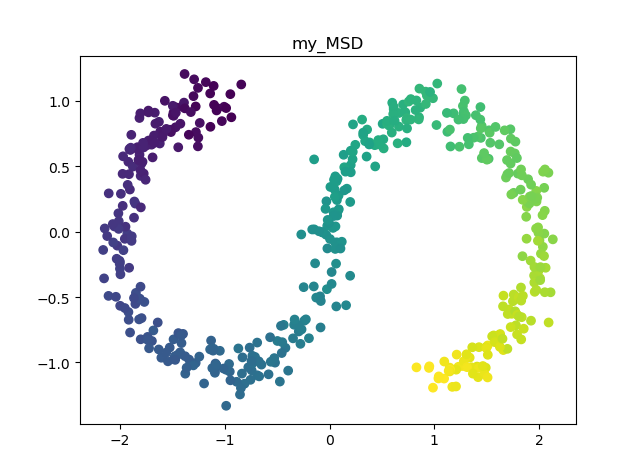
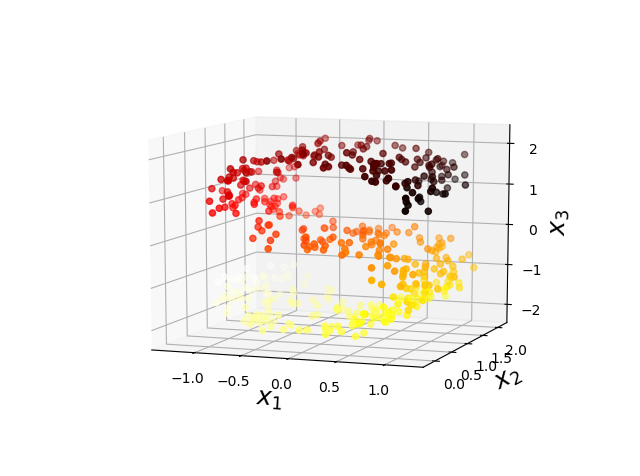


（3）编程实现 MDS 算法，并利用该算法对鸢尾花数据进行降维与可视化，分别利用

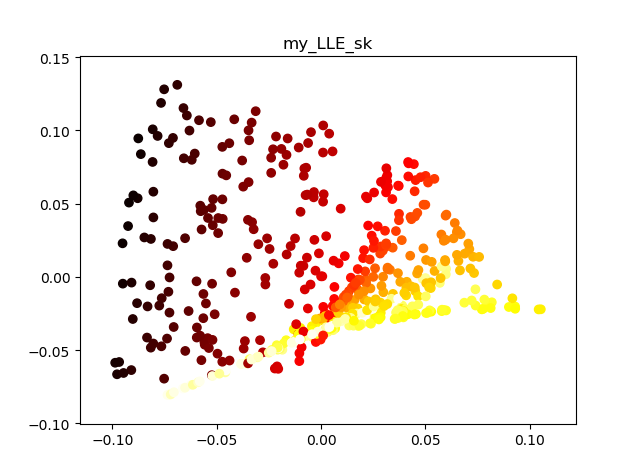
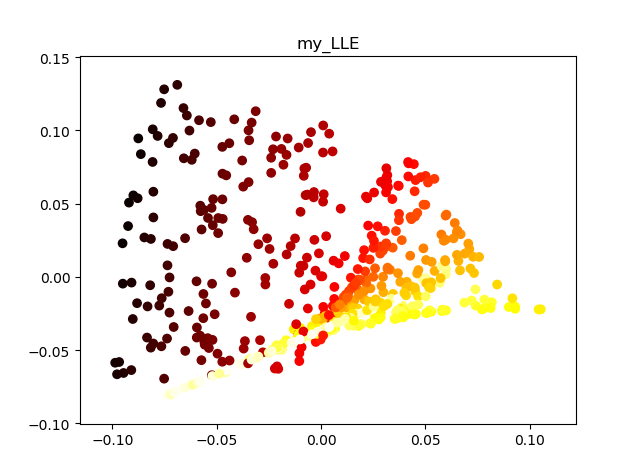
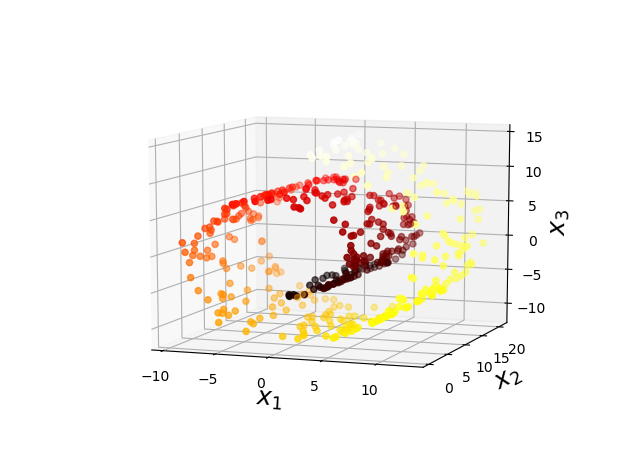
1 范数与 2 范数作为高维距离计算公式，比较降维效果的不同。

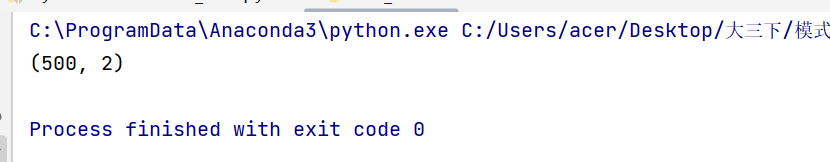


1. 编程实现 ISOMAP 算法，并利用该算法对三维 S 型数据进行降维与可视化。



1. 编程实现 LLE 算法，并利用该算法对瑞士卷数据进行降维与可视化。





2. 对实验结果进行截图，对结果进行必要的解释并说明实验中使用的参数。

3. 写出调试的过程，说明测试用例及调试中遇到的主要问题和解决方法。

4. 写出实验收获与不足，以及对实验的相关意见。

降维作用

（1）使得数据集更容易使用

（2）降低很多算法的计算开销

（3）去除噪声

（4）多维数据不容易画图，降低维度容易画图，使结果容易理解。

优点：降低数据的复杂性，识别出最重要的多个特征。

缺点：不一定需要，有可能损失掉有用信息，仅适用于数值数据。