# 3.7 维度空间

## 3.7.1 流形学习（Manifold learning）

流形学习是一类借鉴了拓扑流形概念的降维方法，“流形”是在局部与欧式空间同胚的空间，即高维输入空间在局部具有欧式空间的性质，能用欧式距离来进行距离计算，这给降维方法带来启发，若低维流形嵌入到高维空间中，则数据样本在高维空间的分布虽然看起来复杂，但在局部上仍具有欧式空间的性质，因此，可以在局部建立降维映射关系，然后将局部映射关系推广至全局，实现降维或不同维度的映射[1]234。流形学习为非线性降维，包括将高维数据映射到低维潜在流形上的各种相关技术算法，其目的是在低维空间中可视化数据，或者学习映射发现数据的内在特征结构，这包括从高维到低维的嵌入，也可为从低维到高维的映射[2, 3]。

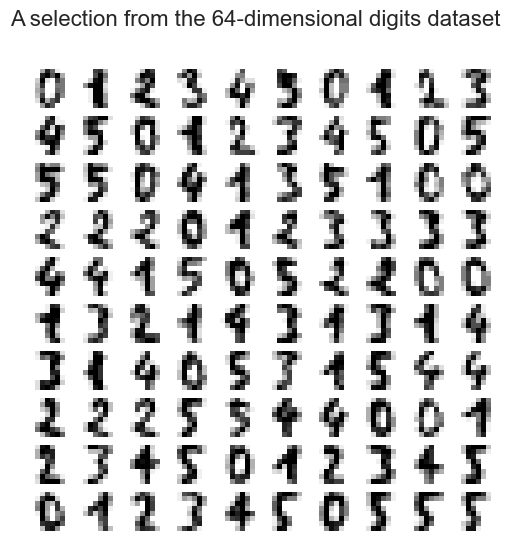
sklearn库的manifold模块集成了Isomap、Locally Linear Embedding、Modified Locally Linear Embedding、Hessian Eigenmapping、Spectral Embedding、Local Tangent Space Alignment、Multi-dimensional Scaling (MDS)和t-distributed Stochastic Neighbor Embedding (t-SNE)等多种算法。

%load\_ext autoreload   
%autoreload 2   
%reload\_ext autoreload  
import usda.manifold as usda\_manifold  
import usda.demo as usda\_demo  
from usda.database import gpd2postSQL,postSQL2gpd  
import usda.utils as usda\_utils  
import usda.data\_visual as usda\_vis  
import usda.pano\_projection\_transformation as usda\_pano  
import usda.models as usda\_model  
  
from sklearn.datasets import load\_digits  
from sklearn import manifold  
from sklearn.datasets import make\_moons,make\_circles  
from matplotlib import pyplot as plt, colors  
  
import numpy as np  
import pandas as pd  
from scipy.linalg import eigh,eig,det  
from numpy import linalg as LA  
  
from scipy.spatial.distance import pdist, squareform  
from sklearn.decomposition import KernelPCA  
import plotly.graph\_objects as go  
  
import os  
import pickle  
from tqdm import tqdm  
import random  
import matplotlib  
  
from sklearn import covariance  
from sklearn import cluster  
import geopandas as gpd  
from matplotlib.collections import LineCollection  
  
import pylab  
import umap.umap\_ as umap  
import plotly.express as px  
from sklearn.cluster import DBSCAN

以sklearn库的[Manifold learning on handwritten digits: Locally Linear Embedding, Isomap…](https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py)①示例为引，查看比较各类嵌入技术在数字数据集上的表现结果。

digits = load\_digits(n\_class=6)  
X, y = digits.data, digits.target  
n\_samples, n\_features = X.shape  
print(X.shape)  
n\_neighbors = 30  
  
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)

(1083, 64)

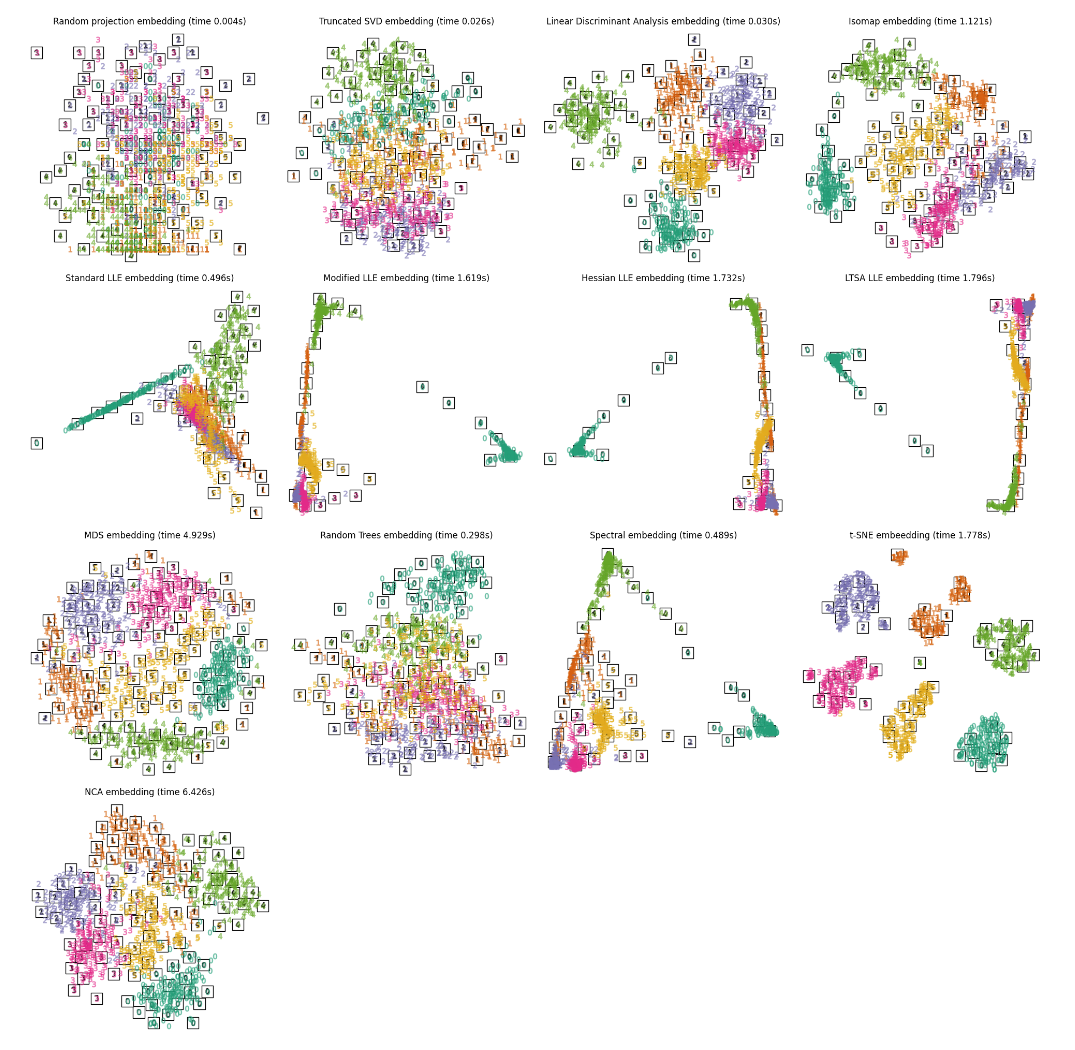


projections,timing,embeddings=usda\_manifold.manifold\_learning\_illustration(X,y)

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...  
Computing Random Trees embedding...  
Computing Spectral embedding...  
Computing t-SNE embeedding...  
Computing NCA embedding...

打印查看 64 维的数字图像映射到 2 维平面空间上的分布。

fig, axes = plt.subplots(4,4,figsize=(20,20))  
  
for idx,name in enumerate(timing):  
 title = f"{name} (time {timing[name]:.3f}s)"  
 ax=axes.flatten()[idx]  
 usda\_manifold.plot\_embedding(projections[name],y,digits,title,ax)  
   
for i in range(16-13):  
 axes.flatten().flat[-(i+1)].set\_visible(False)  
  
fig.tight\_layout()  
plt.show()

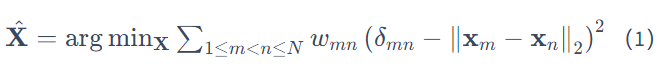


## 3.7.2 MDS，Multidimensional Scaling（多维尺度变换）和 SMACOF（Scaling by MAjorizing a COmplicated Function） 算法[4, 5]

### 3.7.2.1 MDS 算法解析

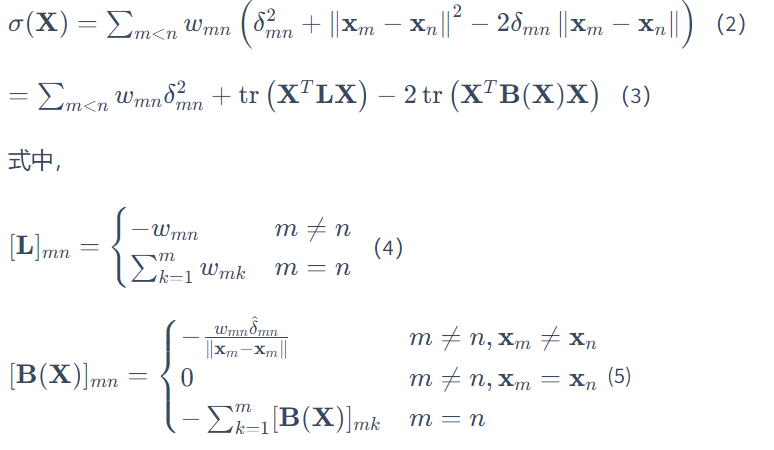
多维尺度变换可以分为度量型（Metric MDS）和非度量型（Non-metric MDS），通常情况下，度量型样本点之间距离为真实距离，例如可以表征空间位置的欧几里得距离；非度量型样本点之间的距离或不相似性通常反应样本点之间的排序而不是真实距离，例如鸢尾花数据特征之间的距离等。关于不同方式的距离度量可以查看*标记距离*部分。

一个数据集有 N 个样本（或图G的顶点/节点），用集合表示为N:={1,…,N}，如果给定样本点之间的两两距离（distances）或不相似性（dissimilarities）{δmn​}(m,n)∈E​，式中E⊆{(m,n)∣1≤m<n≤N}，那么经典 MDS 是寻求 P-维嵌入向量（embedding vectors），嵌入向量X∈RN×P，通过求解以下非凸（non-convex）优化问题进行估计：

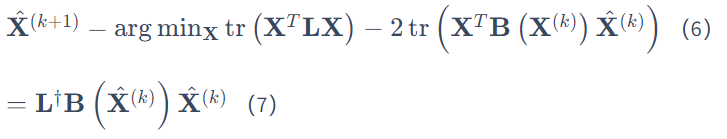


式中wmn​是与δmn​度量相关联的权重，并且如果(m,n)∈/E， wmn​则为0。对于非零的权重，根据应用目的，可以通过多种方式选择，通常简单的设置为1。公式（1）中的目标函数（objective function）称为应力函数（stress function），表示为σ(X)。可以看到，式（1）中得到的最优并不唯一，且表现出平移、旋转和反射的模糊性。

式（1）中的应力最小问题是非凸的，可以用 SMACOF 算法求解到局部最优，展开应力函数得到：



SMACOF 算法的工作原理是用线性函数迭代的最大化式（3）的最后一项，进而相对于X为最小化应力函数。初始为随机值，随后每k次迭代的 SMACOF 更新执行下述公式，

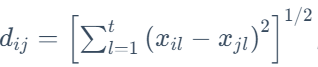
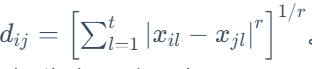


因为B(X)X位于L的值域空间（range space），因此有公式（7）。因为L是秩亏（rank-deﬁcient）的，所以式（6）的解不唯一。然而，当权值{wmn​}指定一个全连接图G:=({1,…,N},E)， L和B(X)有秩N−1，且L的零空间为1，因此式（6）任何解的形式为。此外，如果X(0)初始值以原点为中心，即，式（7）的更新需确保。

sklearn库的[class sklearn.manifold.MDS](https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html)②(n\_components=2, *, metric=True, n\_init=4, max\_iter=300, verbose=0, eps=0.001, n\_jobs=None, random\_state=None, dissimilarity=‘euclidean’, normalized\_stress=‘warn’)实现了 MDS 流形学习（Manifold Learning），*[*sklearn.manifold.smacof*](https://scikit-learn.org/stable/modules/generated/sklearn.manifold.smacof.html)*③(dissimilarities,* , metric=True, n\_components=2, init=None, n\_init=8, n\_jobs=None, max\_iter=300, verbose=0, eps=0.001, random\_state=None, return\_n\_iter=False, normalized\_stress=‘warn’)提供了SMACOF算法，[对应上述公式的核心代码](https://github.com/scikit-learn/scikit-learn/blob/7f9bad99d/sklearn/manifold/_mds.py#L372)④为：

if init is None:  
 # Randomly choose initial configuration  
 X = random\_state.uniform(size=n\_samples \* n\_components)  
 X = X.reshape((n\_samples, n\_components))  
 else:  
 # overrides the parameter p  
 n\_components = init.shape[1]  
 if n\_samples != init.shape[0]:  
 raise ValueError(  
 "init matrix should be of shape (%d, %d)" % (n\_samples, n\_components)  
 )  
 X = init  
  
 old\_stress = None  
 ir = IsotonicRegression()  
 for it in range(max\_iter):  
 # Compute distance and monotonic regression  
 dis = euclidean\_distances(X)  
  
 if metric:  
 disparities = dissimilarities  
 else:  
 dis\_flat = dis.ravel()  
 # dissimilarities with 0 are considered as missing values  
 dis\_flat\_w = dis\_flat[sim\_flat != 0]  
  
 # Compute the disparities using a monotonic regression  
 disparities\_flat = ir.fit\_transform(sim\_flat\_w, dis\_flat\_w)  
 disparities = dis\_flat.copy()  
 disparities[sim\_flat != 0] = disparities\_flat  
 disparities = disparities.reshape((n\_samples, n\_samples))  
 disparities \*= np.sqrt(  
 (n\_samples \* (n\_samples - 1) / 2) / (disparities\*\*2).sum()  
 )  
  
 # Compute stress  
 stress = ((dis.ravel() - disparities.ravel()) \*\* 2).sum() / 2  
 if normalized\_stress:  
 stress = np.sqrt(stress / ((disparities.ravel() \*\* 2).sum() / 2))  
 # Update X using the Guttman transform  
 dis[dis == 0] = 1e-5  
 ratio = disparities / dis  
 B = -ratio  
 B[np.arange(len(B)), np.arange(len(B))] += ratio.sum(axis=1)  
 X = 1.0 / n\_samples \* np.dot(B, X)  
  
 dis = np.sqrt((X\*\*2).sum(axis=1)).sum()  
 if verbose >= 2:  
 print("it: %d, stress %s" % (it, stress))  
 if old\_stress is not None:  
 if (old\_stress - stress / dis) < eps:  
 if verbose:  
 print("breaking at iteration %d with stress %s" % (it, stress))  
 break  
 old\_stress = stress / dis

* 距离计算

MDS方法dissimilarity参数默认计算为euclidean欧几里得距离，如果对于样本点xi​有特征值（坐标值）xi1​,⋯,xit​，那么样本点xi​和xj​之间的欧几里得距离为，在sklearn库中，由sklearn.metrics.pairwise.euclidean\_distances方法计算；MDS 同样适用于一般距离度量函数，例如 MDS 作者 Kruskal, J. B.[6]论文中提到的 Minkowski Lp​， 公式为。如果r≧1.0，该距离函数满足三角不等式成为一个真实的距离函数；当r=2.0时，为欧几里得距离； r=1.0时，该度量为“城市街区距离”或“曼哈顿度量”（Manhattan metric），公式为；当r=∞时，公式变为。

* 单调回归（monotone regression）

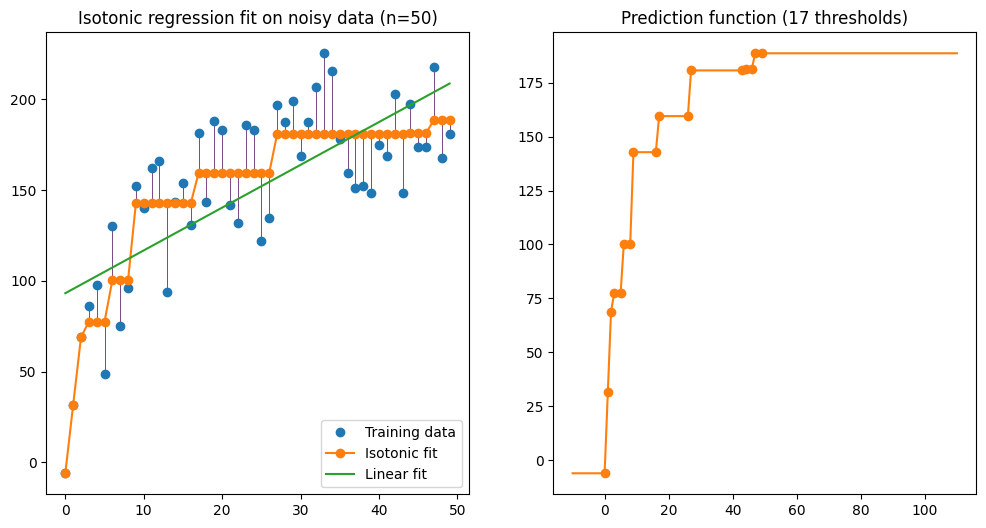
MDS 的核心思想是转化后的嵌入空间（低维空间）样本点之间的距离保持与原始空间（高纬空间）样本点之间的距离一致或尽可能接近，那么最小化应力函数迭代时，嵌入空间样本点间距离的排序，可以通过单调回归，例如代码中的IsotonicRegression()保序回归（Isotonic Regression）方法保持其与原始空间样本点距离（排序）一致。

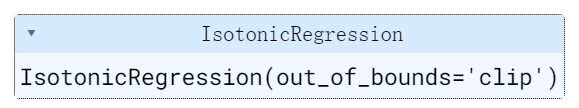
**Isotonic Regression**

Isotonic Regression 是将非递减实（值）函数拟合到一维数据中，求解如下问题：  
，服从当Xi​≤Xj​时， y^​i​≤y^​j​，式中，权重wi​严格为正， x和y为任意实数。Isotonic Regression 依据均方误差由训练数据产生一系列接近目标值y的预测值y^​i​。这些预测值经过插值以预测未知数据，如果超出X的最大和最小值，将以空值替代，因此 Isotonic Regression 的预测形成了一个分段函数。

下述为sklearn库提供的 [Isotonic Regression 示例](https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_isotonic_regression.html)⑤，输入X为一个递增序列，输入在整体递增趋势下局部次序随机变化，当应用 Isotonic Regression 后，预测值​成为一个按照C排序的单调递增序列。

\_,\_,ir=usda\_demo.demo\_isotonic\_regression(n=50)  
ir





### 3.7.2.2 街道空间多维景观指数的 MDS 降维试验

使用*城市空间“语言”试验*视域景观指数和 POI 相关指数计算结果数据，对特征数 12 维的样本数据用 MDS 降维至2维，观察降维后样本点的分布情况。

\_\_C=usda\_utils.AttrDict()  
args=\_\_C  
  
\_\_C.db=usda\_utils.AttrDict()   
\_\_C.db.UN='postgres'  
\_\_C.db.PW='123456'  
\_\_C.db.DB='visual\_metrics'  
\_\_C.db.GC='geometry'   
\_\_C.db.db\_info=dict(geom\_col=args.db.GC,myusername=args.db.UN,mypassword=args.db.PW,mydatabase=args.db.DB)  
  
\_\_C.epsg\_xian=32649

读取数据。

metrics\_inall=postSQL2gpd(table\_name='metrics\_inall',\*\*args.db.db\_info)  
metrics\_inall.sort\_values(by=['fn\_idx'],inplace=True)  
metrics\_inall.dropna(inplace=True)  
metrics\_inall.reset\_index(drop=True,inplace=True)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is metrics\_inall.

MDS 降维。

fields=['Green view index','Sky view factor', 'Ground view index','Equilibrium degree', 'Perimeter area ratio(mn)','Shape index(mn)', 'Fractal dimension(mn)', 'Color richness index','Key point size(0-10]', 'Key point size(10-20]','Key point size(30-40]', 'Key point size(20-30]',]  
metrics\_selection\_df=metrics\_inall[fields]  
  
embedding=manifold.MDS(n\_components=2, normalized\_stress='auto')  
X\_transformed=embedding.fit\_transform(metrics\_selection\_df.to\_numpy())  
  
metrics\_inall[['embedding\_x','embedding\_y']]=pd.DataFrame(X\_transformed)  
# metrics\_inall.to\_crs(args.epsg\_xian,inplace=True)  
metrics\_inall[['lon','lat']]=metrics\_inall.apply(lambda row:[row.geometry.x,row.geometry.y],axis=1,result\_type='expand')  
metrics\_inall.tail(2)

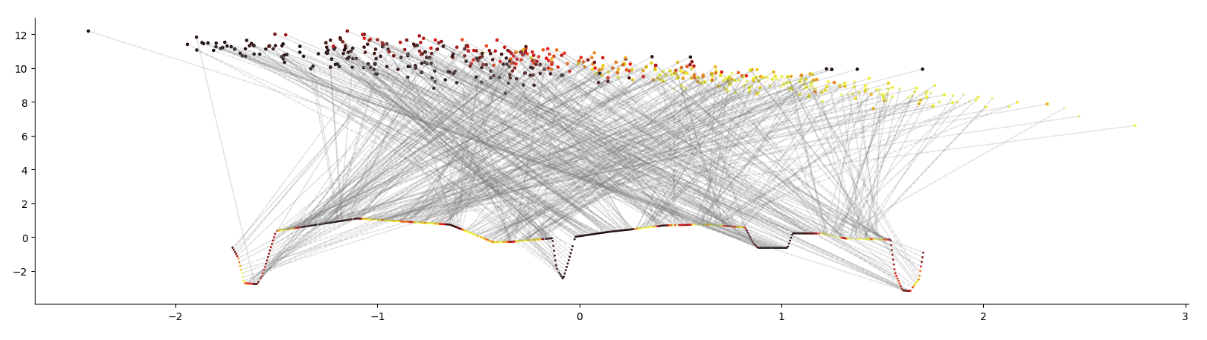
|  | **Green view index** | **Sky view factor** | **Ground view index** | **Equilibrium degree** | **number of patches** | **Perimeter area ratio(mn)** | **Shape index(mn)** | **Fractal dimension(mn)** | **Color richness index** | **Key point size(0-10]** | **...** | **fn\_key** | **fn\_idx** | **geometry** | **frank\_e** | **num** | **num\_diff** | **embedding\_x** | **embedding\_y** | **lon** | **lat** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | 8.876533 | 34.406317 | 49.708967 | 49.21693 | 223.0 | 248.194856 | 1.249436 | 1.033388 | 58.475149 | 205 | ... | dongxistreet | 579 | POINT (108.96774 34.26069) | 64.362740 | 41 | -1.0 | -19.157652 | 95.471806 | 108.967744 | 34.260691 |
| **564** | 8.703033 | 33.874200 | 50.756650 | 49.15724 | 188.0 | 270.015482 | 1.202361 | 1.026970 | 64.774178 | 146 | ... | dongxistreet | 580 | POINT (108.96779 34.26075) | 63.841717 | 42 | -34.0 | -63.913861 | 138.190518 | 108.967795 | 34.260749 |

2 rows × 24 columns

通过指数聚类贡献度计算得知绿视率（Green view index，GVI）分值最高，因此数据打印时，样本点颜色表示为 GVI 数值。同时，折线部分为按样本点实际地理位置坐标标准化后相对位置打印结果；折现上部的散点为 MDS 降维至2维后值的标准化相对位置。降维后嵌入空间（2维）样本点的两两距离应该接近于原空间（12 维）样本点的两两距离，这从结果打印中也能够观察到，具有相似值的 GVI 样本点在嵌入空间中彼此相互靠近成簇。

norm\_lon=usda\_utils.normalize\_by\_meanNstd(metrics\_inall[['lon']])[0]  
norm\_lat=usda\_utils.normalize\_by\_meanNstd(metrics\_inall[['lat']])[0]  
  
norm\_x=usda\_utils.normalize\_by\_meanNstd(metrics\_inall[['embedding\_x']])[0]  
norm\_y=usda\_utils.normalize\_by\_meanNstd(metrics\_inall[['embedding\_y']])[0]+10

fig,ax=plt.subplots(figsize=(20,5))  
  
color=usda\_vis.vals4color\_cmap(metrics\_inall['Green view index'].values)  
ax.scatter(norm\_lon,norm\_lat,s=1,color=color)  
ax.scatter(norm\_x,norm\_y,s=5,color=color,)  
  
for idx,(s\_lon,s\_lat,e\_x,e\_y) in pd.concat([norm\_lon,norm\_lat,norm\_x,norm\_y],axis=1).iterrows():  
 dx=e\_x-s\_lon  
 dy=e\_y-s\_lat  
 ax.arrow(s\_lon,s\_lat,dx,dy,width=0.001,head\_width=0.01,head\_length=0.01,color='gray',alpha=0.2,linestyle='--')  
   
ax.spines['top'].set\_visible(False)  
ax.spines['right'].set\_visible(False)  
plt.show()



## 3.7.3 KPCA（Kernel to PCA（Principal Component Analysis））（核主成分分析）

### 3.7.3.1 特征值和特征向量几何意义图解

* 特征值和特征向量

在流形学习诸多方法中均涉及到线性变换，将向量左乘变换矩阵，使得该向量的长度和方向发生变化，投射到一个新的空间位置。KPCA 流形学习方法则应用到线性变换（映射）下特征向量和特征值的计算。对于线性变换可以查看*编程线性代数*部分，这里进一步图示特征值和特征向量的空间几何意义，有助于对流形学习各类算法的理解。

在线性代数中，线性变换的特征向量（eigenvector，characteristic vector）是一个非零向量，对应每一特征向量存在一个标量，即特征值（eigenvalue），用λ表示，例如对变换矩阵A=，用numpy或SciPy库提供的工具计算特征值和特征向量，得到特征值为1和3， 对应特征值的特征向量为[-0.707,0.70710678]和[0.707,0.707]（为矩阵的列， eigenvectors[:, i]）。在几何中，变换矩阵旋转、拉伸、或剪切它作用的向量，而特征值是拉伸特征向量的因子。如果特征值为负，则反方向拉伸。

定义：设A是n阶矩阵，如果数λ和n维非零列向量x使关系式Ax=λx成立，那么，这样的数λ称为矩阵A的特征值，非零向量x称为A对应于特征值λ的特征向量[7]117。A @ x == eigvals[1] \* x结果为真，验证了该定义关系式Ax=λx，即转换矩阵A把特征向量x转化成它自身的标倍量（scalar multiple）λ。

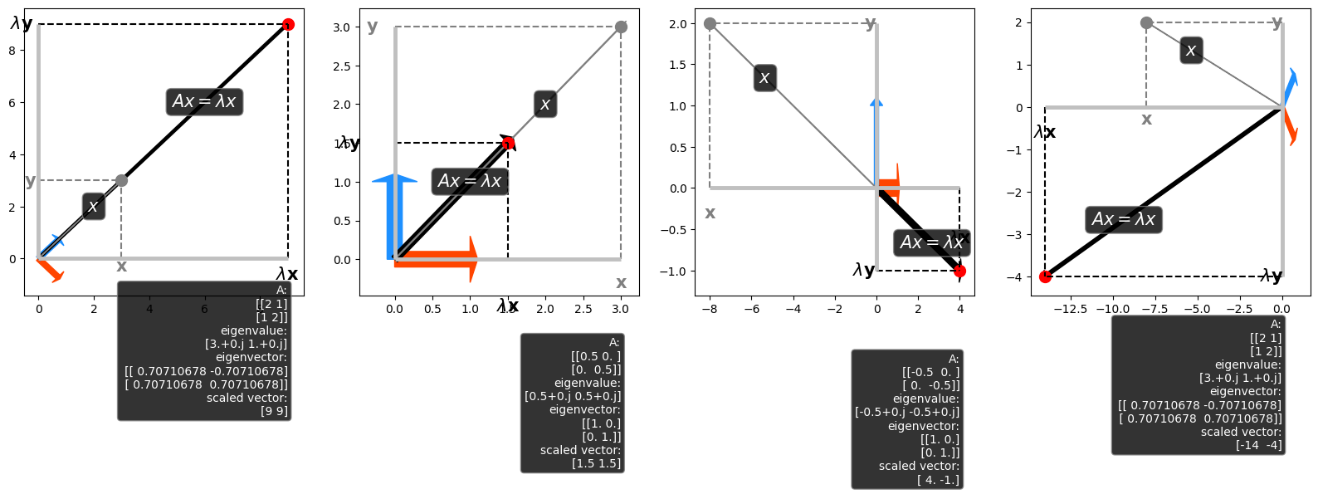
A=np.array([[2,1],[1,2]])  
eigvals, eigvecs=LA.eigh(A)  
print(eigvals, '\n',eigvecs)  
  
print('-'\*50)  
x=eigvecs[:,1].reshape(2,1)  
print(A @ x == eigvals[1] \* x)

[1. 3.]   
 [[-0.70710678 0.70710678]  
 [ 0.70710678 0.70710678]]  
--------------------------------------------------  
[[ True]  
 [ True]]

* 如果为实数（real）且。如下图左1，将向量[3,3]，拉伸了3倍，变为向量[9，9]；
* 如果如下图左2，将向量[3,3]，缩小了0.5倍，变为向量[1.5,1.5]；
* 如果效果缩放大小同前2条，但是方向取反。如下图左3，将向量[-8,2]，缩小了0.5倍，变为向量[4,-1][8]451；

如果向量不位于特征向量的方向上，例如下图右1向量[-8,2]，经转换矩阵的缩放同时，也发生旋转，变为向量[-14,-4]。

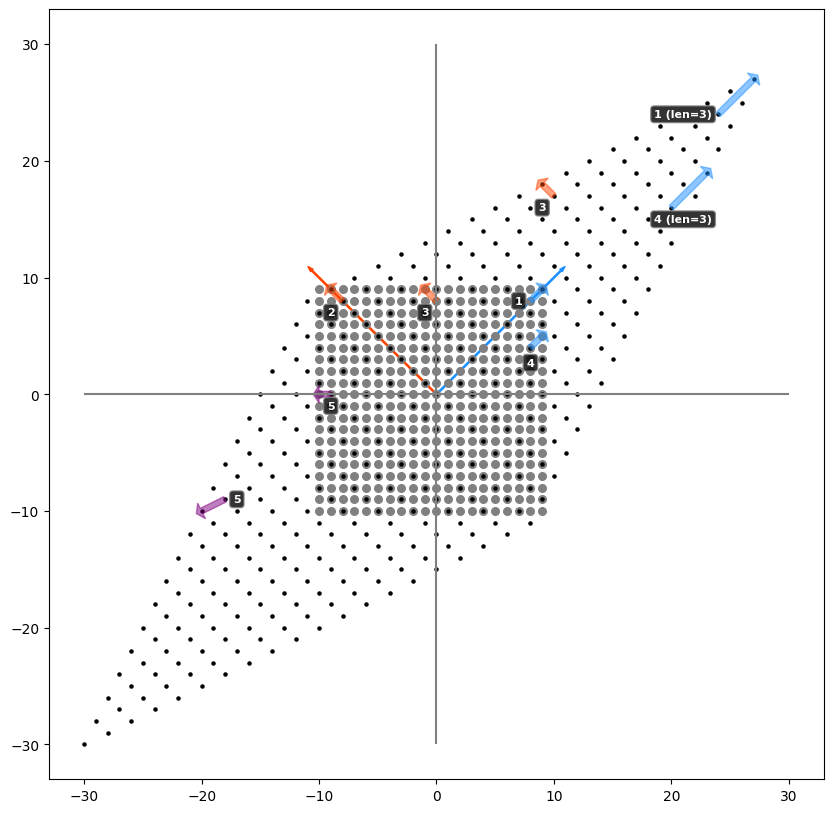
fig, axs=plt.subplots(1, 4,figsize=(20,4.5))  
usda\_demo.demo\_eigvals\_eigvecs\_1d(np.array([[2,1],[1,2]]) ,np.array([3,3]),ax=axs[0])   
usda\_demo.demo\_eigvals\_eigvecs\_1d(np.array([[0.5,0],[0,0.5]]) ,np.array([3,3]),ax=axs[1])  
usda\_demo.demo\_eigvals\_eigvecs\_1d(np.array([[-0.5,0],[0,-0.5]]) ,np.array([-8,2]),ax=axs[2])  
usda\_demo.demo\_eigvals\_eigvecs\_1d(np.array([[2,1],[1,2]]) ,np.array([-8,2]),ax=axs[3])  
plt.show()



类似上述单个向量（1维）的变换，同时变换向量矩阵（2维）观察图像变化，同样用变换矩阵*A*=试验。从下图得知，位于2个特征向量方向上的点沿所属方向缩放，如点1和4，点2（重合）和3；点5不位于任一特征向量方向上，因此除了缩放，也发生了旋转。

usda\_demo.demo\_eigvals\_eigvecs\_2d()

A:  
[[2 1]  
 [1 2]]  
eigenvalue:  
[1. 3.]  
eigenvector:  
[[-0.70710678 0.70710678]  
 [ 0.70710678 0.70710678]]



* 对角化和特征分解（Diagonalization and the eigendecomposition）

假设变换矩阵A的特征向量构成一组基向量（Base Vector），即A有n个线性无关的特征向量v1​,v2​,⋯,vn​，对应的特征值为λ1​,λ2​,⋯,λn​。定义一个方阵Q，其列向量是A的n个线性无关的特征向量， Q=[v1​​v2​​⋯​vn​​]；因为Q的每一个列都是A的特征向量，因此AQ=[λ1​v1​​λ2​v2​​⋯​λn​vn​​]；定义一个对角矩阵Λ，其中每一个对角元素Λij​是与Q第i列相关联的特征值，有AQ=QΛ；因为Q的列向量线性无关，所以Q是可逆的，等式两边同时乘以Q有A=QAQ−1，或者为Q−1AQ=Λ。

因此，变换矩阵可以分解成一个由它的特征向量组成的矩阵和一个特征值沿着对角线的对角矩阵，及特征向量矩阵的逆矩阵，即为特征分解。本质上，变换矩阵A和对角矩阵Λ为用两个不同的基表示的同一线性变换。当将线性变换表示为Λ时，特征向量被用作基[9]。

用SciPy库的eig方法分解变换矩阵，并用U @ np.diag(D) @ np.linalg.inv(U)验证了特征分解的结果。

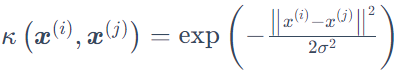
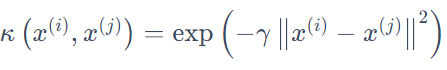
A=np.array([[2,1],[1,2]])  
D,U=eig(A)  
print(D,'\n',U,'\n','-'\*50)  
print(U @ np.diag(D) @ np.linalg.inv(U))

[3.+0.j 1.+0.j]   
 [[ 0.70710678 -0.70710678]  
 [ 0.70710678 0.70710678]]   
 --------------------------------------------------  
[[2.+0.j 1.+0.j]  
 [1.+0.j 2.+0.j]]

### 3.7.3.2 核函数（kernel function）与核技巧（kernel trick）[10]

核函数定义，设X是输入空间（input space）（欧式空间Rn的子集或离散集合），又设H为特征空间（feature space）（希尔伯特空间（Hilbert space）），如果存在一个从X到H的映射ϕ(x):X→H，使得对所有x,z∈X，函数K(x,z)满足条件K(x,z)=ϕ(x)⋅ϕ(z)，则称K(x,z)为核函数， ϕ(x)为映射函数，式中ϕ(x)⋅ϕ(z)为ϕ(x)和ϕ(z)的内积。

核技巧的想法是，在学习与预测中只定义核函数K(x,z)，而不显示地定义映射函数ϕ，通常，直接计算K(x,z)比较容易，而通过ϕ(x)和ϕ(z)计算K(x,z)并不容易实现。ϕ是输入空间mathbfRn到特征空间H的映射，特征空间H一般为高维空间，甚至无穷维。对于给定的核K(x,z)，特征空间H和映射函数ϕ的取法并不唯一，可以取不同的特征空间，即便同一特征空间里也可以取不同的映射。

通常核函数是正定核函数（positive definite kernel function），常用的核函数有多项式核函数（polynomial kernel function）、高斯核函数（Gaussian kernel function）（或径向基函数（radial basis function，RBF））和双曲正切核函数（hyperbolic tangent (sigmoid) kernel function）等。多项式核函数公式为κ(x(i),x(j))=(x(i)Tx(j)+θ)p，式中，θ为阈值， p为指数值，均由用户指定；RBF 公式为，引入变量，则公式书写为；双曲正切核函数公式为。

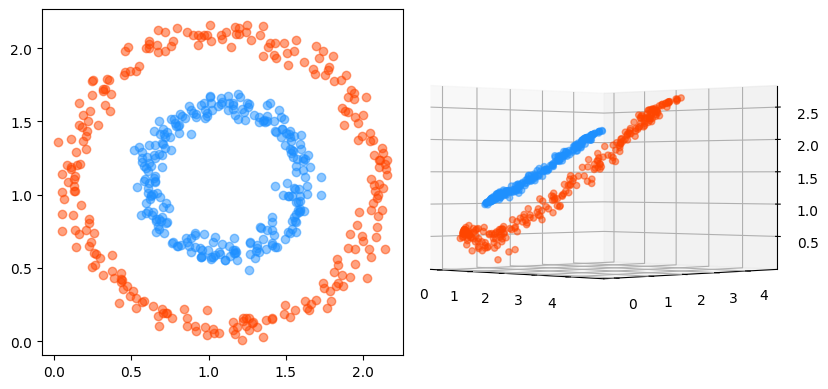
### 3.7.3.3 KPCA[11][12]169-179算法解析

KPCA 是基于 PCA 的一种非线性形式，将的样本（解释变量）数据通过核函数（Kernel function）映射到一个更高维度的特征空间（feature space）(记为）中实现类的线性可分，再应用 PCA 方法降维。

* 映射

将样本数据x∈Rk映射到d维子空间上，定义非线性映射函数ϕ:Rk→Rd(d≫k)。例如，样本集X为包含2个特征（k=2）的列向量，将其映射到3维空间中可以表述为：。如下图对应的2维输入空间样本到3维特征空间的映射结果，在高维特征空间实现样本类的线性可分，表面映射函数ϕ能够处理非线性问题。

usda\_demo.demo\_KPCA\_2d\_to\_3d\_mapping()



* 特征空间的特征向量

定义原始输入空间样本X=[x1​,x2​,…,xn​]，每一列表示一个样本，每个样本点xi​为K维列向量， X中共N个样本，为K×N大小矩阵。将X用映射函数ϕ映射到高维特征空间F中，得到一个D×N大小的新矩阵ϕ(X)=[ϕ(x1​),ϕ(x2​),…,ϕ(xn​)]。

在 PCA 中，两个特征k和j之间的协方差为σjk​=n1​∑i=1n​(xj(i)​−μj​)(xk(i)​−μk​)；标准中心化样本特征，使得μj​=0和μk​=0，简化协方差公式为σjk​=n1​∑i=1n​xj(i)​xk(i)​，即为样本的点积（内积）；表述为一般形式Σ=n1​∑i=1n​x(i)x(i)T=n1​XTX。

由上述公式，映射到特征空间的样本协方差为Σ=n1​∑i=1n​ϕ(x(i))ϕ(x(i))T=n1​ϕ(X)Tϕ(X)，为一个D×D的矩阵，为获得特征向量（主成分），从协方差矩阵推导公式有：

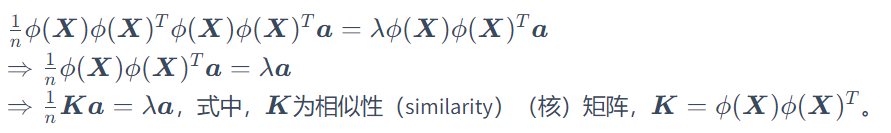
Σv=λv⇒n1​∑i=1n​ϕ(x(i))ϕ(x(i))Tv=λv⇒n1​ϕ(X)Tϕ(X)v=λv（1），式中，λ和v为协方差矩阵Σ的特征值和特征向量。因为没有显示的定义映射函数ϕ，无法计算ϕ(x(i))ϕ(x(i))T，因此（1）式无法直接求解，需要借助核技巧解决。

* 应用核函数与核技巧

继续推导（1）式有  
，因为式中为标量，因此有，式中a为N维列向量，α=[α1​,α2​,…,αn​]T；

将（2）式带回（1）式得1n1​ϕ(X)Tϕ(X)ϕ(X)Ta=λϕ(X)Ta（3）；

将（3）式等号两侧均乘以*ϕ*(***X***)得：



* KPCA 降维的代码实现

1. 计算输入空间样本的欧几里德距离的平方，用ScipPy库的pdistn维空间中观测值之间的成对距离计算方法实现，对应代码为

sq\_dists = pdist(X, 'sqeuclidean')

1. 将两两距离转换成一个方阵，对应代码为

mat\_sq\_dists = squareform(sq\_dists)

1. 用高斯核函数计算核矩阵，对应代码为

K = exp(-gamma \* mat\_sq\_dists)

1. 使用公式K′=K−1n​K−K1n​+1n​K1n​，中心化K，使用公式对应代码为

N = K.shape[0]  
one\_n = np.ones((N,N)) / N  
K = K - one\_n.dot(K) - K.dot(one\_n) + one\_n.dot(K).dot(one\_n)

1. 计算K的特征向量和特征值，对应代码为

eigvals, eigvecs = eigh(K)  
eigvals, eigvecs = eigvals[::-1], eigvecs[:, ::-1]

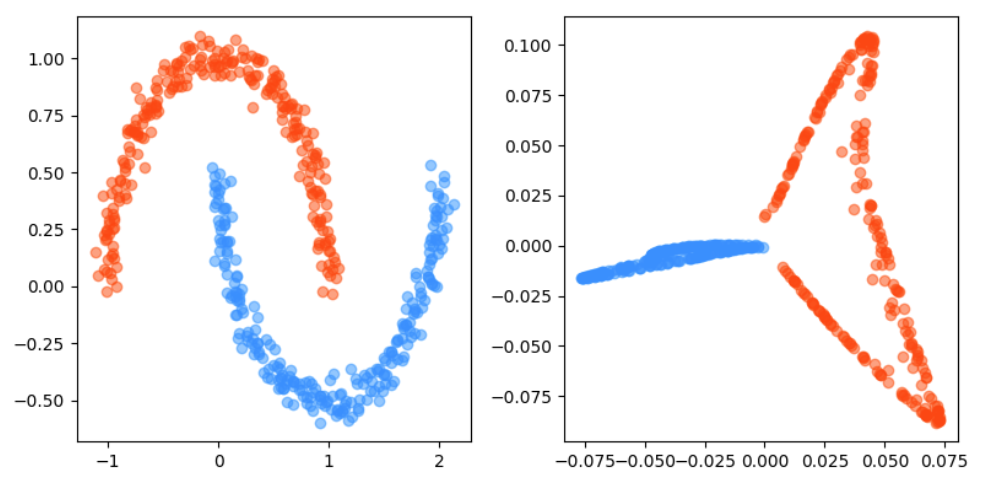
1. 返回前个特征向量，对应代码为

X\_pc = np.column\_stack([eigvecs[:, i] for i in range(n\_components)])

试验数据计算结果如下。

X, y = make\_moons(n\_samples=500,noise=0.05)  
X\_kpca,eigvals,eigvecs,K=usda\_manifold.rbf\_kernel\_pca(X, gamma=15, n\_components=2)  
print(eigvals.shape,eigvecs.shape,K.shape)  
  
fig,axes=plt.subplots(1,2,figsize=(8,4))  
axes[0].scatter(X[y==0, 0], X[y==0, 1],color='orangered', marker='o', alpha=0.5)  
axes[0].scatter(X[y==1, 0], X[y==1, 1],color='dodgerblue', marker='o', alpha=0.5)  
  
axes[1].scatter(X\_kpca[y==0, 0], X\_kpca[y==0, 1], color='orangered', marker='o', alpha=0.5)  
axes[1].scatter(X\_kpca[y==1, 0], X\_kpca[y==1, 1],color='dodgerblue', marker='o', alpha=0.5)  
plt.tight\_layout()  
plt.show()

(500,) (500, 500) (500, 500)



### 3.7.3.4 街道空间单一景观指数的 KPCA 映射试验

同样使用*城市空间“语言”试验*视域景观指数和 POI 相关指数计算结果数据，但仅选择 POI 均衡度frank\_e一个指数，通过将该一个维度的样本数据映射到特征空间后，再降维到2个维度，观察样本点线性可分的分离状态。KPCA 方式的维度变换结果可以辅助观察样本点自身属性特征的集聚分布结构，有助于对城市某一属性的空间分布特征做出判断。

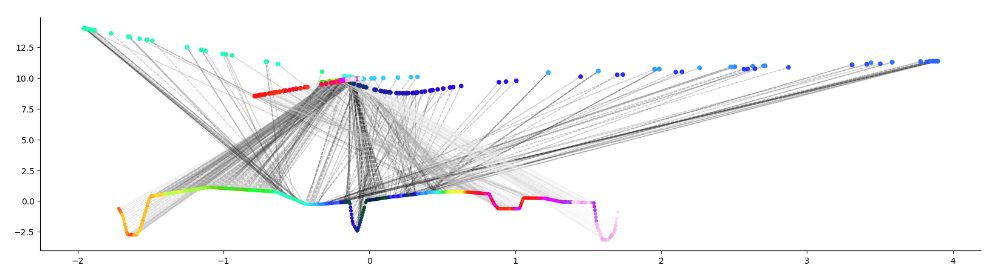
使用sklearn库提供的KernelPCA方法计算，计算时如果配置n\_components=None，返回值将为特征向量的所有方向。

fields=['frank\_e']  
metrics\_selection\_df=metrics\_inall[fields]  
  
kernel\_pca=KernelPCA(n\_components=None, kernel="rbf", gamma=15, fit\_inverse\_transform=True, alpha=0.1)  
kernel\_pca=kernel\_pca.fit\_transform(metrics\_selection\_df.to\_numpy())  
print(kernel\_pca.shape)  
kernel\_pca\_df=pd.DataFrame(kernel\_pca[:,:2],columns=['x','y'])  
print(kernel\_pca\_df.shape)  
  
norm\_x=usda\_utils.normalize\_by\_meanNstd(kernel\_pca\_df[['x']])[0]  
norm\_y=usda\_utils.normalize\_by\_meanNstd(kernel\_pca\_df[['y']])[0]+10  
  
kernel\_pca\_df[['lon','lat']]=metrics\_inall.apply(lambda row:[row.geometry.x,row.geometry.y],axis=1,result\_type='expand')  
norm\_lon=usda\_utils.normalize\_by\_meanNstd(kernel\_pca\_df[['lon']])[0]  
norm\_lat=usda\_utils.normalize\_by\_meanNstd(kernel\_pca\_df[['lat']])[0]

(565, 249)  
(565, 2)

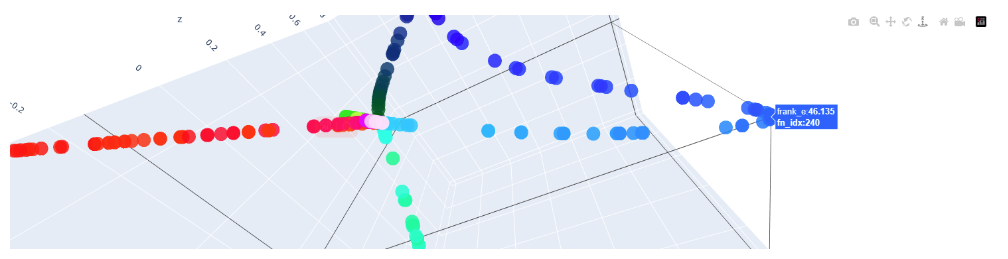
下述可视化图表样本点间的边也表述了均衡度值的大小，浅色的值小，表明该区域 POI 行业服务类型单一；深色的值大，表明该区域 POI 行业服务类型信息熵高，各类行业服务类型丰富均衡混杂。从 KPCA 映射结果中可以明显的分割成几段区域，代表样本点各自区域特征的显著性。

fig,ax=plt.subplots(figsize=(20,5))  
  
color4edges=usda\_vis.vals4color\_cmap(metrics\_selection\_df['frank\_e'].values,cmap='gist\_gray',vmin=metrics\_selection\_df['frank\_e'].min(), vmax=metrics\_selection\_df['frank\_e'].max())  
for idx,(s\_lon,s\_lat,e\_x,e\_y) in pd.concat([norm\_lon,norm\_lat,norm\_x,norm\_y],axis=1).iterrows():  
 dx=e\_x-s\_lon  
 dy=e\_y-s\_lat  
 ax.arrow(s\_lon,s\_lat,dx,dy,width=0.001,head\_width=0.01,head\_length=0.01,color=color4edges[idx],alpha=0.2,linestyle='--')  
  
color=usda\_vis.vals4color\_cmap(metrics\_selection\_df['frank\_e'].values,cmap='gist\_ncar',vmin=metrics\_selection\_df['frank\_e'].min(), vmax=metrics\_selection\_df['frank\_e'].max())  
ax.scatter(norm\_lon,norm\_lat,s=10,color=color)  
ax.scatter(norm\_x,norm\_y,s=20,color=color,)  
   
ax.spines['top'].set\_visible(False)  
ax.spines['right'].set\_visible(False)  
plt.show()



提取前3个主成分，在三维空间中查看数据点的分布。

fig = go.Figure(data=[go.Scatter3d(  
 x=kernel\_pca[:,0],  
 y=kernel\_pca[:,1],  
 z=kernel\_pca[:,2],  
 mode='markers',  
 hovertemplate='frank\_e:<b>%{customdata[0]:.3f}</b><br><b>fn\_idx:%{customdata[1]}</b><extra></extra>',  
 customdata=metrics\_inall[['frank\_e','fn\_idx']],  
 marker=dict(  
 size=12,  
 color=color,   
 opacity=0.8,   
 )  
)])  
  
# tight layout  
fig.update\_layout(margin=dict(l=0, r=0, b=0, t=0))  
fig.show()



## 3.7.4 Isomap

### 3.7.4.1 Isomap 算法解析

在研究全球气候模式、空间分布模式特征时，通常要寻找隐藏在高维观测中有意义的低维结构。Tenenbaum, J. B.等人[13]提出了 Isomap（isometric feature mappin） 降维的方法，该方法使用易于度量的局部度量信息（local metric information）来学习数据集潜在的全局几何结构（global geometry）。与经典的主成分分析（PCA）和多维尺度变换（MDS）等经典技术不同，Isomap 可以发现复杂的自然观测背后的非线性自由度，且有效的计算全局最优解，对重要的一类数据流形，保证渐近收敛到真实结构。

从下述 Isomap 试验生成的3维数据（“瑞士卷”，Swiss roll）可以观察到，两点间的最近距离不是直线距离，而是通过测地线（geodesic）或最短路径（shortest path）测量的距离。将其路径结果映射在2维平面上，获得了反应了流形的真实低维几何形状。距离度量上分为两步，第一步是根据指定的邻近点数量或者半径提取每个样本点的邻近点集合（可用sklearn.neighbors.NearestNeighbors方法计算），及计算样本点到各邻近点的距离（例如 欧几里德距离）（可用sklearn.metrics.pairwise\_distances方法计算），即局部度量信息；第二步是根据样本点的邻近点距离矩阵，计算所有样本点两两间最短距离矩阵（可用scipy.sparse.csgraph.shortest\_path方法计算），一般通常用 Floyd-Warshall 算法、 Dijkstra 算法，Bellman-Ford 算法或 Johnson 算法等，即全局几何结构。

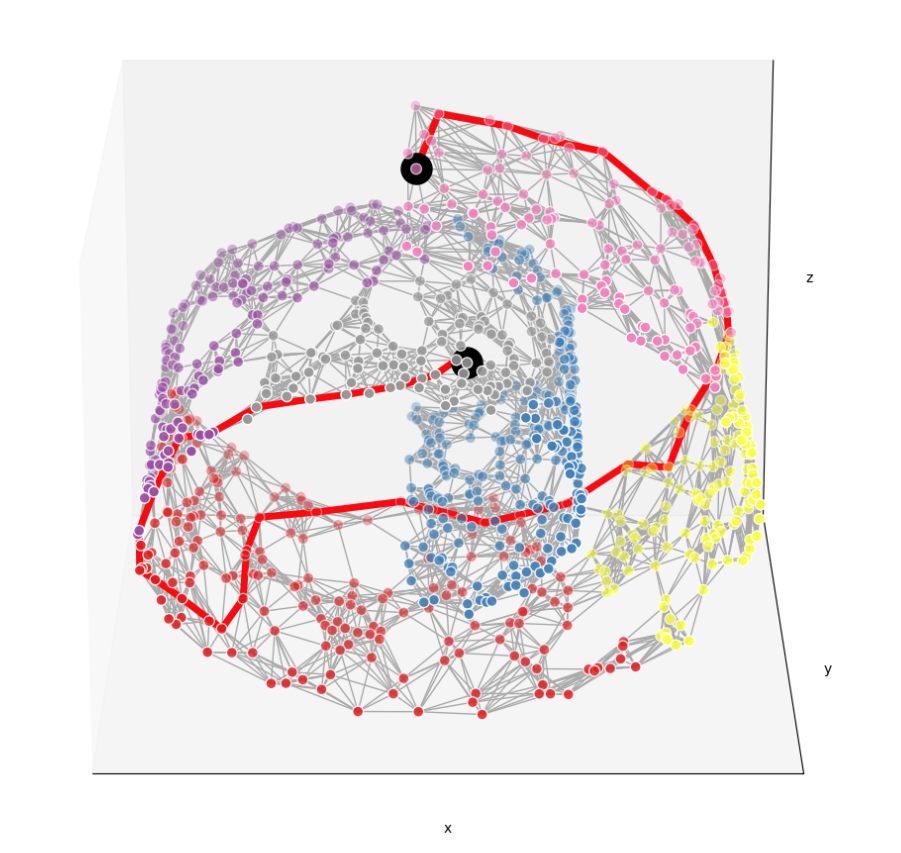
获得了样本点两两间距离矩阵后，就可以用 MDS 或者 KPCA 等流形学习方法降维，例如sklearn提供的Isomap方法使用了 KPCA 的方式。

d\_isomap=usda\_demo.Demo\_Isomap(n\_components=2)   
d\_isomap.agglomerative\_clustering()  
d\_isomap.isomap\_G()

Compute structured hierarchical clustering...  
Elapsed time: 0.06s  
Number of points: 1000

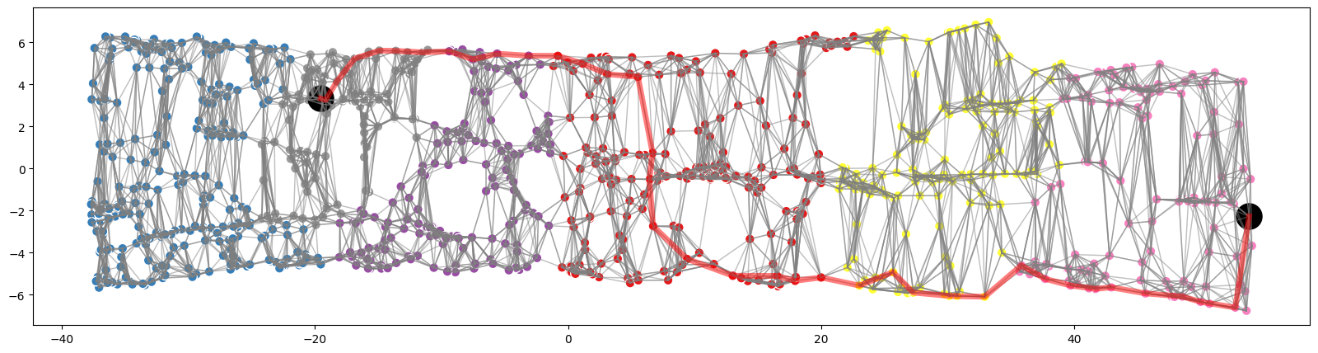
查看试验生成的三维原始数据，及示范的一个两点间的最短路径。

d\_isomap.plot\_3d()



用 Isomap 降维至2维，样本点间的距离保持了高维空间中的真实距离。

d\_isomap.plot\_2d()



### 3.7.4.2 街道视域全景语义分割图的 Isomap 降维和嵌入空间的主成分分布

在前文的试验中是使用由街道视域全景语义分割图计算的各类视域景观指数和 POI 相关指数作为输入数据。此次则直接使用全景语义分割图作为数据输入，用 Isomap 降至2维后查看样本点的分布，并根据样本点的位置嵌入分割图像，查看图像在嵌入空间主成分的分布情况。

读取*城市空间“语言”试验*中生成的全景分割图标签数据。

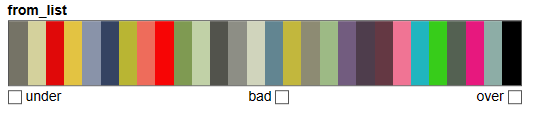
seg\_label\_root=r'G:\data\pano\_dongxistreet\pano\_seg\seg\_label'  
fns=os.listdir(seg\_label\_root)  
data\_lst=[]  
for i, fn in tqdm(enumerate(fns)):  
 with open(os.path.join(seg\_label\_root,fn),'rb') as f:  
 seg\_label=pickle.load(f)  
 data\_lst.append(seg\_label)  
   
random.shuffle(data\_lst)   
data=np.stack(data\_lst)   
print(data.shape)  
print(data[0])

566it [00:00, 1488.49it/s]  
  
(566, 512, 1024)  
[[22 22 22 ... 22 22 22]  
 [22 22 22 ... 22 22 22]  
 [22 22 22 ... 22 22 22]  
 ...  
 [27 27 27 ... 27 27 27]  
 [27 27 27 ... 27 27 27]  
 [27 27 27 ... 27 27 27]]

​

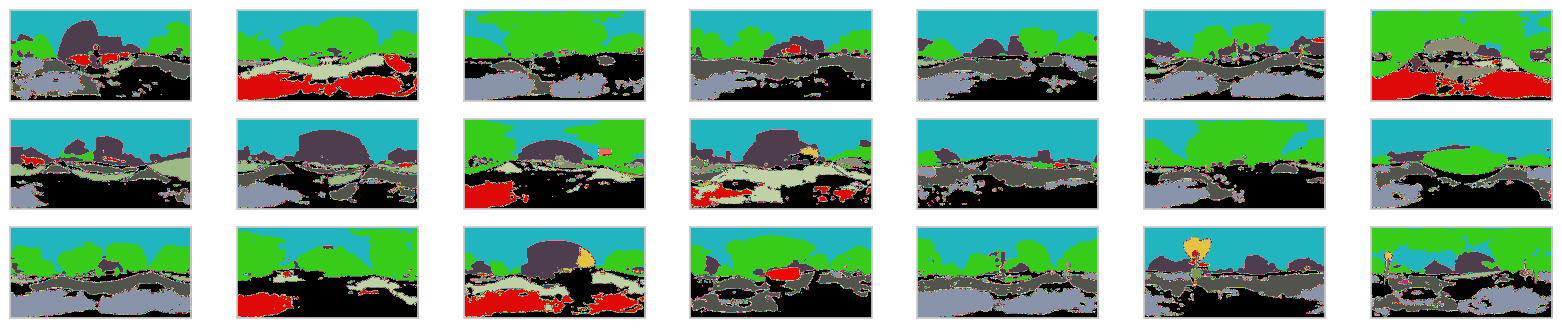
定义对应语义分割图标签的配色方案。

levels=list(usda\_pano.label\_color\_name.keys())+[28]  
colors=[[i/255 for i in v[0]] for v in usda\_pano.label\_color\_name.values()]  
cmap, norm = matplotlib.colors.from\_levels\_and\_colors(levels,colors)  
cmap



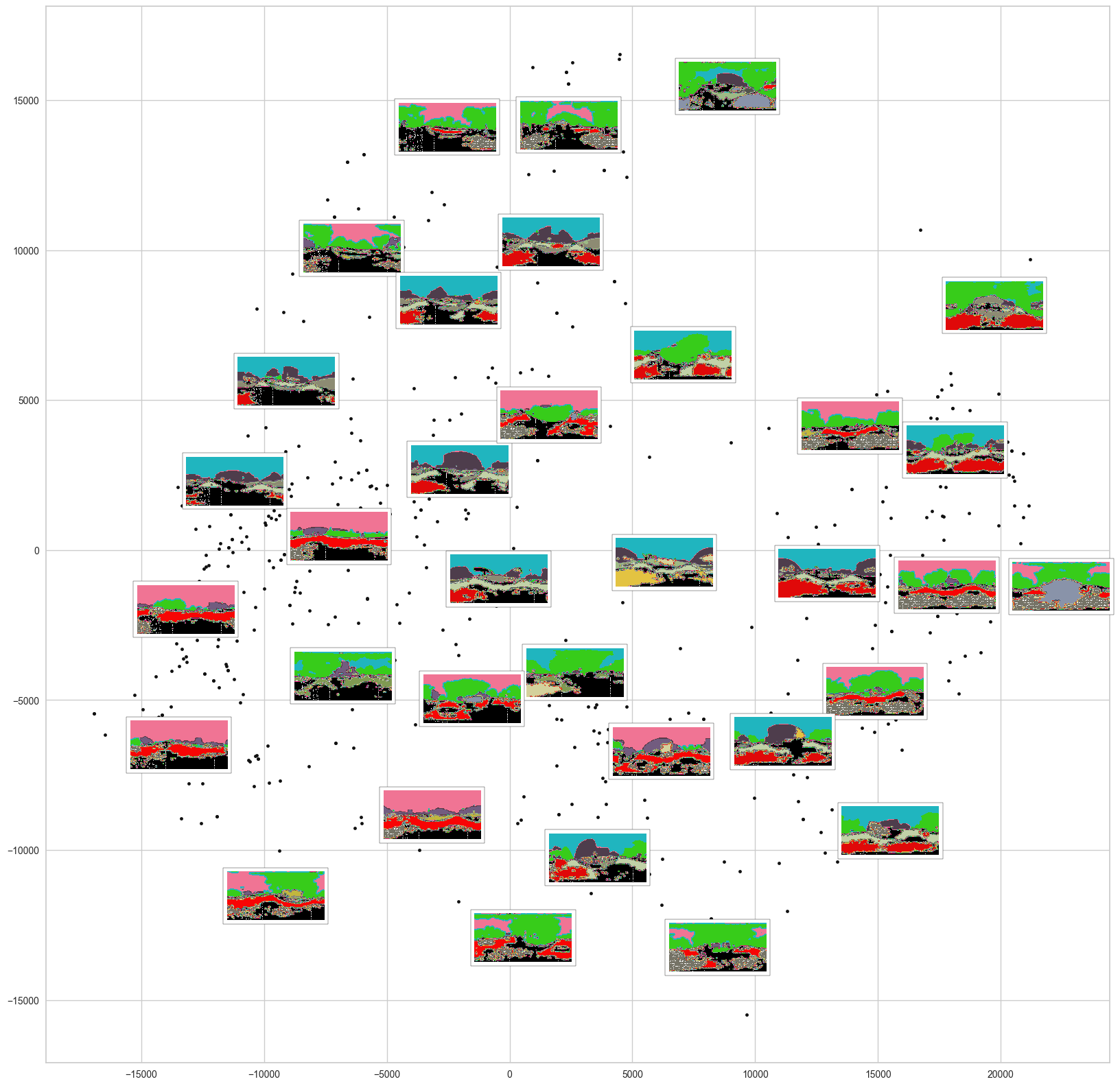
打印查看部分全景语义分割图。

fig, ax = plt.subplots(3, 7, subplot\_kw=dict(xticks=[], yticks=[]), figsize=(20,4))  
for i, axi in enumerate(ax.flat):  
 \_ = axi.imshow(data[i], cmap=cmap);



定义plot\_components()方法，传入流形学习模型（本例为 Isomap），将语义分割图嵌入到主成分分布位置，方便可视化分析不同样本点区域分割图像的特征。从分布结果来看，样本点的 Isomap 降维保留了高维空间样本点之间的距离关系（流形结构），可以根据降维后样本点的分布结构，应用聚类等手段推断提取样本的特征分类。

fig, ax=plt.subplots(figsize=(20, 20))  
data\_=data.reshape(data.shape[0],-1)  
usda\_model.plot\_components(data\_, model=manifold.Isomap(n\_components=2, n\_neighbors=8), images=data, thumb\_frac=0.1,cmap=cmap,ax=ax,img\_scale=0.1)

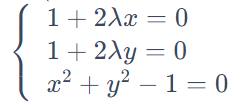


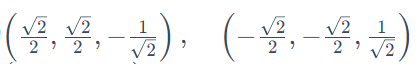
## 3.7.5 LLE，Locally Linear Embedding（局部线性嵌入）

### 3.7.5.1 拉格朗日乘数/乘子法（Lagrange multiplier）

在数学优化问题中，拉格朗日乘子法（以数学家 Joseph-Louis Lagrange 命名）是一种寻找受一个或多个条件（方程）约束的（多元）函数局部最大值和最小值，即极值的策略。这种方法将一个有n个变量与k个约束条件的最优化问题转化为一个有n+k个变量的方程组极值问题，且其变量不再受任何约束。该方法由条件引入的新未知标量称为拉格朗日乘数，从而找到能让设出的隐函数微分为零的未知数的值。该方法概况如下：为找到受g(x)=0条件约束下f(x)函数的最大或最小值，构造拉格朗日函数（Lagrangian function），L(x,λ)≡f(x)+λ⋅g(x)，并找到以x和λ为函数L的驻点（stationary points ），即极值点，该函数的所有偏导数为零，有 并 ，相当于。

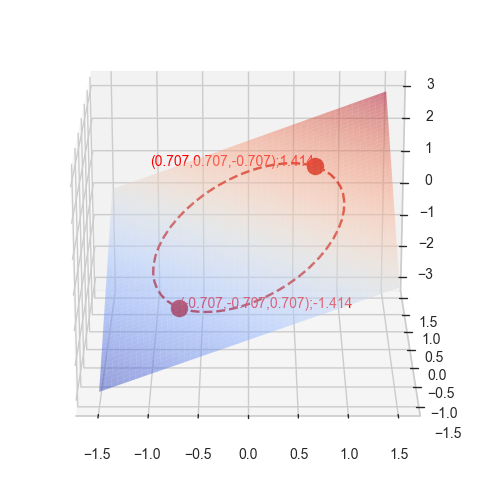
例如，最大化函数f(x,y)=x+y，并服从。由条件方程得出，因此拉格朗日函数为，当g(x,y)为0时，等价于f(x,y)。

计算梯度有：，得出，，式中，最后一个方程为条件约束。由前两个方程得出，将其结果代入最后一个方程得到，因此​。

由上述计算得出L的驻点为，代入目标函数f(x,y)，得到，即服从约束条件下的最大和最小值。

将该案例写入为代码，对应的公式为x + y + L \* (x\*\*2 + y\*\*2 - 1)，用SciPy的fsolve方法解算，示例结果如下。

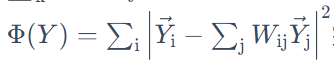
usda\_demo.demo\_lagrange\_multiplier\_xy3D()



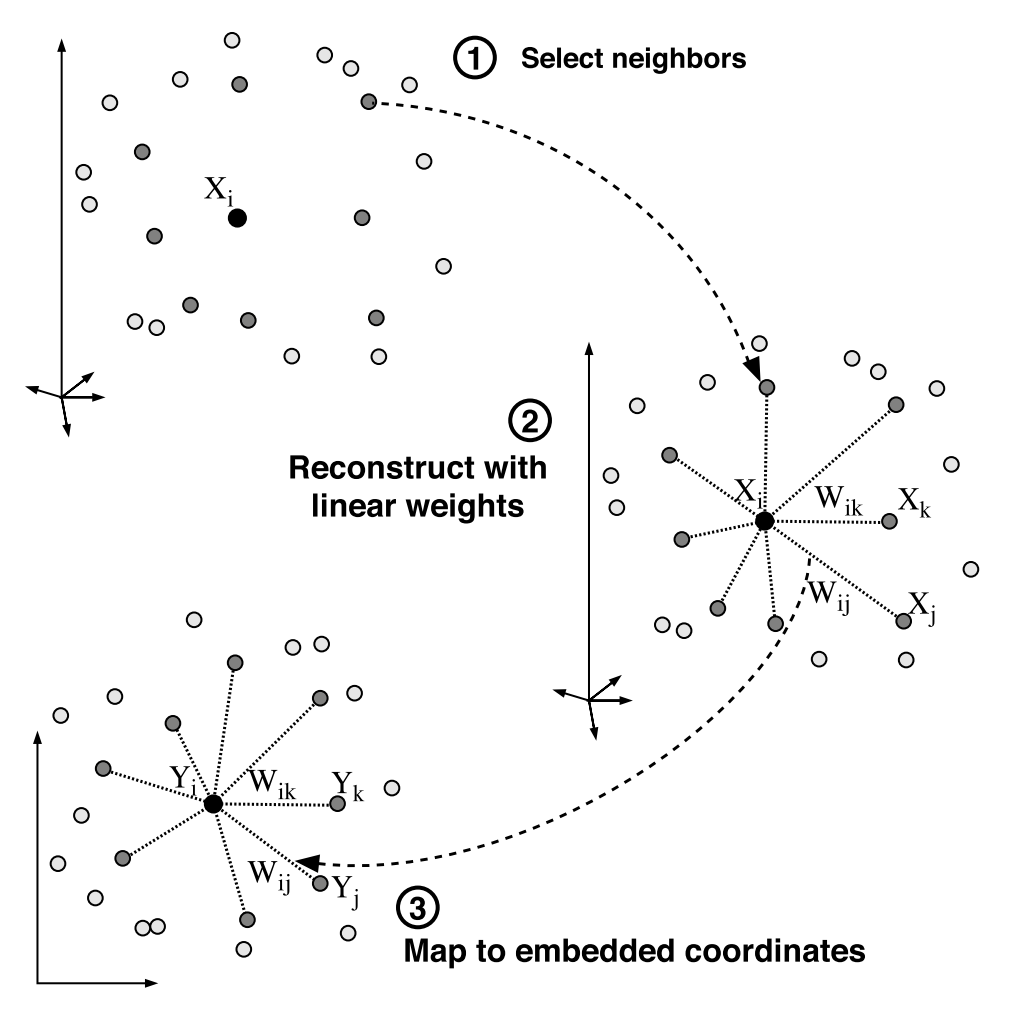
### 3.7.5.2 LLE 算法解析

许多科学领域依赖于探索性数据分析和可视化，因此对大量多元高维的数据分析提出了降维的需求，即找到高维数据的紧凑表示（compact representations）。Roweis, S. T.等人[15]提出了局部线性嵌入（locally linear embedding，LLE），一种无监督学习算法，用于降维计算。与局部降维的聚类方法不同，LLE 将其高维输入映射到一个低维的全局坐标系中，且其优化不涉及局部最小值。利用线性重构的局部对称性，LLE 能够学习非线性流形的全局结构。

下图[15]为 LLE 算法的图解，其计算主要分为3个步骤，

1. 计算每一个数据点​的近邻点，例如使用K-近邻模型（k-nearest neighbors algorithm，k-NN），可以指定近邻点数量 ；
2. 由的近邻点线性组合加权重构，通过求解均方误差损失函数（目标函数）计算权重Wij​；
3. 计算由*W*ij​重构的低维嵌入向量​，通过寻找*M*ij​=*δ*ij​−*W*ij​−*W*ji​+∑k​*W*ki​*W*kj​ 中稀疏对称矩阵的的最小本征模（eigenmodes）来最小化完成。。

LLE 的基本思想是，在低维嵌入空间中使用与高维输入空间相同的构造权值。虽然权重Wij​和向量Yi​是用线性代数的方法计算得到，但是由数据点的近邻点线性重构的约束将导致高度非线性的嵌入。

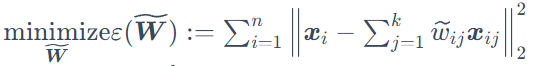


对 LLE 算法的详细推导过程引用 Ghojogh, B.[16]等人对 LLE 综述的论文。

#### 1) -近邻

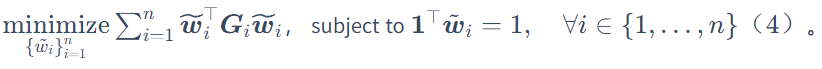
利用数据点两两之间的欧式距离构建数据点的 kNN 最近邻网络，其中每个数据点有k个邻近点， xij​∈Rd表示数据点xi​的第j个近邻点；并且矩阵包含xi​的k个近邻数据点。

#### 2）由近邻点线性重构

根据每个数据点的近邻点求出其线性重构的权值。在高维输入空间中，该线性重构的目标函数（表达式）为 ，式中，  
包括权值，为第i个数据点的第k个邻近点线性重构的权值，且xij​∈Rd为第i个数据点的第j个邻近点。

约束条件表示每个数据点的线性重构权值之和为1。注意，某些权值可能为负，且非常大的正负权值可以相互抵消，也使得权值总和为1，这一事实会导致某些权值爆炸的问题。然而，因为优化问题的解有一个封闭形式，权值爆炸的问题不会发生。如果迭代求解，则权重值会积累增长并可能导致爆炸。目标函数可重写为。

约束条件意味着，因此，，简化目标函数有，，式中， Gi​为格拉姆矩阵（gram matrix），定义为

最终，公式（1）可以写为：

对公式（4）利用拉格朗日乘子法，得到

将拉格朗日函数的导数设为0，给出，

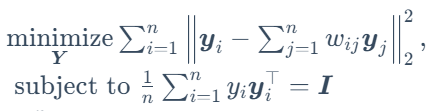
由公式（5）和（6）得出，

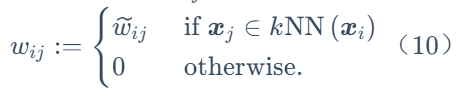
由公式（5）和（7）得出，

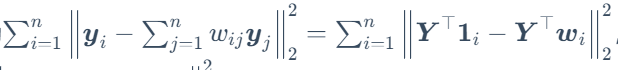
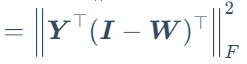
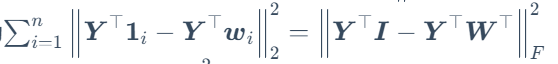
由公式（3）可知，矩阵Gi​∈Rk×k的秩最多等于min(k,d)。如果d<k， Gi​是奇异的，且Gi​应该被替换为Gi​+ϵI，式中， ϵ是一个很小的正数。通常数据是高维的（k≪d)），例如图像数据，因此如果Gi​是全秩的，对它进行反向不会有任何问题。这种对G主对角线的强化在 LLE 中称为正则化。这种数值方法广泛应用于流形和子空间的学习中。

3）线性嵌入

在上一步高维输入空间找到了局部数据点线性重构的权值，在低维嵌入空间使用与输入空间相同的权值来嵌入数据，这种线性嵌入可以表述为以下优化问题：



，式中， ***I***为单位矩阵。为按行堆叠嵌入的数据点， ***y****i*​∈R*p*是第*i*个嵌入的数据点，且*wij*​是来自于高维输入空间线性重构的权值，为*xi*​近邻点*xj*​的权值，非近邻点的权值全部配置为0， 公式（9）中的第2个约束条件确保嵌入数据点的均值为0。第1个和第2个约束共同满足嵌入数据点有单位协方差（unit covariance）。

假设，，且为第*i*个元素为1，其它元素值均为0的向量。公式（9）目标函数可以重述为​，用矩阵形式表示为，式中，的第i行包含第i个数据点的所有权值； 表示矩阵的弗罗贝尼乌斯（Frobenius）范数。公式（11）简化为：   
，式中tr(.)表示矩阵的迹，且。  
注意到，由公式（1），矩阵W的列加和为1可知 (I−W)是矩阵W的拉普拉斯式（Laplacian）。因此根据公式（13），矩阵M能够看作权值矩阵拉普拉斯式的格拉姆矩阵（gram matrix）。

最终，公式（9）可以写为,  subject to ，式中，1和0的维数分别为Rn和Rp。公式（14）可以理解为输入数据X和嵌入数据Y之间依赖关系的最大化。如果忽略第2个约束条件，公式（14）的拉格朗日函数为：，式中，Λ∈Rn×n是一个包含拉格朗日乘子的对角矩阵。将拉格朗日函数的导数设为0，给出，这是M的特征向量问题。因此Y的列是M的特征向量，其中特征值为的对角元素。

由于公式（14）是一个最小化问题，所以Y的列应该按照从小到大的特征值进行排序。此外， M中的 (I−W)是权值W的拉普拉斯矩阵。在线性代数和图论中，如果一个图有k个不相交的连通部分，那么它的拉普拉斯矩阵有k个零特征值。因为kNN图，或者W是一个连通图， (I−W)有一个零特征值，其特征向量为1=[1,1,…,1]⊤。在对特征向量按从最小到最大的特征值进行排序后，需要忽略第一个特征值为零的特征向量，取M的p个最小非零特征值的特征向量作为Y∈Rn×p的列。

### 3.7.5.3 街道 POI 行业类别空间分布结构

[Visualizing the stock market structure](https://scikit-learn.org/stable/auto_examples/applications/plot_stock_market.html#sphx-glr-auto-examples-applications-plot-stock-market-py)⑥使用了几种无监督学习技术从历史报价的变化中提取股票市场结构。将这一方法应用到街道空间 POI 行业类别相关性结构分析中，可以得到街道业态分布的关系结构。

应用*城市空间“语言”试验*部分处理过的 POI 数据，并用定义的street\_poi\_structure()方法提取街道每一个位置点，给定半径（500m）内 POI 数据按一级分类各类别的数量， 进而求出每一位置点各个类别占当前位置点总 POI 数量的比例。

\_\_C=usda\_utils.AttrDict()  
args=\_\_C  
  
\_\_C.db=usda\_utils.AttrDict()   
\_\_C.db.UN='postgres'  
\_\_C.db.PW='123456'  
\_\_C.db.DB='visual\_metrics'  
\_\_C.db.GC='geometry'   
\_\_C.db.db\_info=dict(geom\_col=args.db.GC,myusername=args.db.UN,mypassword=args.db.PW,mydatabase=args.db.DB)

clipped\_pois\_fn=r'G:\data\POI\_dongxistreet\clipped\_poi\_gdf.gpkg'  
poi\_gdf=gpd.read\_file(clipped\_pois\_fn)  
metrics\_merge\_gdf=postSQL2gpd(table\_name='metrics\_merge',\*\*args.db.db\_info)  
coordi\_df=metrics\_merge\_gdf.to\_crs(poi\_gdf.crs)  
  
pos\_poi\_idxes\_gdf,pos\_poi\_feature\_vector\_gdf=usda\_pano.street\_poi\_structure(poi=poi\_gdf,position=coordi\_df,distance=500)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is metrics\_merge.  
  
  
100%|██████████| 566/566 [01:36<00:00, 5.84it/s]

gpd2postSQL(pos\_poi\_feature\_vector\_gdf,table\_name='pos\_poi\_feature\_vector',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is pos\_poi\_feature\_vector.

写入数据库，方便调用。

pos\_poi\_feature\_vector\_gdf=postSQL2gpd(table\_name='pos\_poi\_feature\_vector',\*\*args.db.db\_info)  
pos\_poi\_feature\_vector\_gdf.tail(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is pos\_poi\_feature\_vector.

|  | **geometry** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **...** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | POINT (309138.206 3793057.251) | 7.0 | 4.0 | 1.0 | 1.0 | 13.0 | 15.0 | 11.0 | 4.0 | 6.0 | ... | 1.0 | 10.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **564** | POINT (309146.205 3793057.353) | 7.0 | 5.0 | 1.0 | 1.0 | 13.0 | 15.0 | 10.0 | 4.0 | 6.0 | ... | 2.0 | 10.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **565** | POINT (309154.205 3793057.456) | 7.0 | 5.0 | 1.0 | 1.0 | 13.0 | 14.0 | 10.0 | 4.0 | 6.0 | ... | 2.0 | 11.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

3 rows × 26 columns

查看 POI 数据一级分类标签对应的整数编码。

poi\_classificationName=usda\_pano.poi\_classificationName  
print(poi\_classificationName)

{0: 'delicacy', 1: 'hotel', 2: 'shopping', 3: 'lifeService', 4: 'beauty', 5: 'spot', 6: 'entertainment', 7: 'sports', 8: 'education', 9: 'media', 10: 'medicalTreatment', 11: 'carService', 12: 'trafficFacilities', 13: 'finance', 14: 'realEstate', 15: 'corporation', 16: 'government', 17: 'entrance', 18: 'naturalFeatures', 19: 'administrativeLandmarks', 20: 'address', 21: 'road', 22: 'busStop', 23: 'subwayStop', 24: 'businessDistrict'}

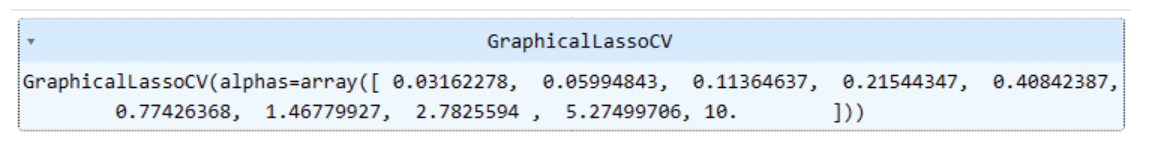
计算每一位置点一级分类各类别数量占比，并除以其标准差归一化处理。

cols=[str(i) for i in list(poi\_classificationName.keys())]  
poi\_num=pos\_poi\_feature\_vector\_gdf[cols].to\_numpy()+1  
X=poi\_num/poi\_num.sum(axis=1)[:,None]  
X /= X.std(axis=0)  
print(X.shape)

(566, 25)

协方差矩阵（covariance matrix）的逆矩阵，通常称为精度矩阵（precision matrix），与偏相关矩阵（partial correlation matrix）成正比，其给出了部分独立关系，即如果两个特征在其它特征上是条件独立的，则精度矩阵中相应的系数为零。这就是为什么估计稀疏精度矩阵是有意义的：通过学习数据的独立关系为条件，更好的估计协方差矩阵。这被称为协方差选择（covariance selection）。在小样本情况下，即参数n\_samples是参数n\_features的数量级或更小，稀疏逆协方差估计器（sparse inverse covariance estimators ）往往比收缩协方差估计器（shrunk covariance estimators）工作得更好。然而，在相反的情况下，如果各个特征间相关性很高，则会引起数值上的不稳地。另外，与收缩协方差估计器不同，稀疏逆协方差估计器能够恢复非对角结构。GraphicalLasso方法使用惩罚加强精度矩阵的稀疏性，参数alpha值越高，精度矩阵越稀疏；相应的GraphicalLassoCV方法可使用交叉验证自动配置alpha参数[17]。

alphas=np.logspace(-1.5, 1, num=10) # Return numbers spaced evenly on a log scale.  
edge\_model=covariance.GraphicalLassoCV(alphas=alphas)   
edge\_model.fit(X)



print(f'---covariance matrix:\n{edge\_model.covariance\_[:2]}\n---precision matrix:\n{edge\_model.precision\_[:2]}')

---covariance matrix:  
[[ 1.00000000e+00 2.52449861e-01 -8.13242161e-03 -1.12667801e-06  
 2.71112537e-02 -1.20745417e-01 4.40653861e-02 1.95542545e-02  
 -3.84202137e-02 5.09840049e-03 -3.52740955e-01 -6.08977938e-03  
 0.00000000e+00 -3.47908633e-01 -1.51052575e-01 -5.73324093e-07  
 -1.91494601e-02 -1.26217666e-02 -1.26218600e-02 -1.26219251e-02  
 -1.26219703e-02 -1.26219046e-02 -1.26219803e-02 -1.26220302e-02  
 -1.26220924e-02]  
 [ 2.52449861e-01 1.00000000e+00 -3.70717527e-02 -2.93142231e-06  
 1.70702429e-02 -2.78792475e-01 7.89998860e-02 1.20174350e-02  
 -4.36633276e-02 2.26520909e-02 -1.14570509e-01 -2.38152757e-02  
 0.00000000e+00 -2.60774625e-01 -2.82061911e-01 -1.49169064e-06  
 -1.78193503e-02 -4.76883404e-02 -4.76883300e-02 -4.76883404e-02  
 -4.76883352e-02 -4.76879429e-02 -4.76881171e-02 -4.76881866e-02  
 -4.76882980e-02]]  
---precision matrix:  
[[ 1.31552604 -0.17091593 0. 0. -0. 0.  
 -0. -0.01253992 0. -0. 0.3892473 0.  
 -0. 0.34799309 0.09109158 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. ]  
 [-0.17091593 1.25557047 0. 0. -0. 0.27579507  
 -0. -0. 0. -0. 0. 0.  
 -0. 0.19774892 0.28442421 0. 0. 0.00980012  
 0.00980036 0.00980035 0.00980026 0.00979957 0.00980007 0.00980028  
 0.00980056]]

使用 AP（Affinity Propagation）聚类反应街道所有位置点各个 POI 一级分类组成比例关系的协方差矩阵，自动计算聚类簇数量，获得一级行业分类在整条街道上具有类似影响的关系。

\_, labels = cluster.affinity\_propagation(edge\_model.covariance\_, random\_state=0)  
n\_labels = labels.max()  
labels

array([0, 0, 5, 1, 4, 3, 0, 5, 4, 4, 3, 5, 2, 3, 3, 1, 4, 5, 5, 5, 5, 5,  
 5, 5, 5], dtype=int64)

查看各簇对应一级行业分类名的结果。

poi\_names=np.array(list(poi\_classificationName.values()))  
for i in range(n\_labels + 1):  
 print(f"Cluster {i + 1}: {', '.join(poi\_names[labels == i])}")

Cluster 1: delicacy, hotel, entertainment  
Cluster 2: lifeService, corporation  
Cluster 3: trafficFacilities  
Cluster 4: spot, medicalTreatment, finance, realEstate  
Cluster 5: beauty, education, media, government  
Cluster 6: shopping, sports, carService, entrance, naturalFeatures, administrativeLandmarks, address, road, busStop, subwayStop, businessDistrict

出于可视化目的，使用 LLE 流形学习，以一级行业分类为“样本”，以各个位置对应分类的频数比例为特征，即高维输入空间数据，降至2维，即低维嵌入空间。用LocallyLinearEmbedding方法计算。

node\_position\_model = manifold.LocallyLinearEmbedding(n\_components=2, eigen\_solver="dense", n\_neighbors=7)  
embedding = node\_position\_model.fit\_transform(X.T).T

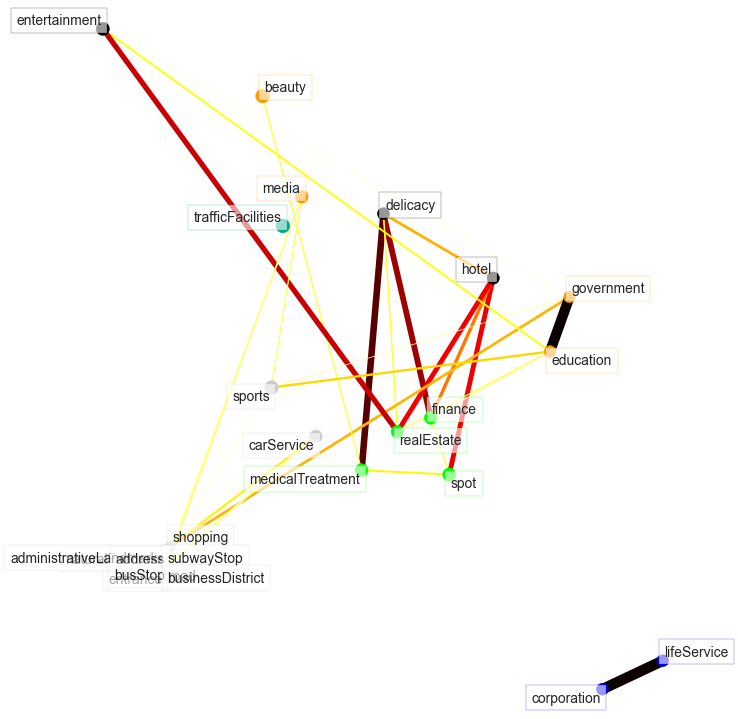
打印最终网络，其中顶点代表一级行业分类，同时，

* 聚类标签颜色用于定义顶点颜色；
* 稀疏协方差模型用于显示边的强度；
* 2维嵌入向量用于定位顶点位置。

从结果可以得知，对该街道的业态分布，

* 在聚类结果上，delicacy，hotel 和 entertainment 在街道业态组成上具有相似影响，其它各簇同；
* 在基于精度矩阵的一级行业类别相关性上，government 和 education 具有较高的关联，其次为 delicacy 和 medical treatment 及 finance等；
* 对 LLE 流形学习结果，shopping，subway stop，business district，bus stop，administrtive landmarks等等集聚在一起，表面这些业态在所有位置点的 POI 组成比例上相似。其它相互邻近的点意义同。

usda\_manifold.partial\_correlations\_embedding2Dgraph(embedding,edge\_model,labels,poi\_names,figsize=(7,7))

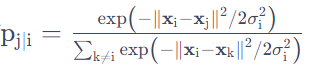


## 3.7.6 t-SNE 和 UMAP

### 3.7.6.1 t-SNE 算法解析

Hinton, G. E.等人[18]描述了一种概率方法，称为 SNE（Stochastic Neighbor Embedding），在保持样本点邻里关系条件下将由高维输入空间向量或两两不相似度（pairwise dissimilarities）描述的对象映射到低维嵌入空间中。以高维输入空间各个样本点为中心的高斯密度定义该样本点与所有样本点的概率分布。嵌入的目的是映射到低维嵌入空间的对象与高维输入空间对象尽可能具有相似的分布。衡量低维嵌入空间和高维输入空间对象分布的差异则使用 Kullback-Leibler 散度（divergences）/距离，计算各个对应样本点之间的距离和为最小作为目标函数，并应用梯度调整低维嵌入空间样本点的位置。基于 SNE，van der Maaten 等人[19]提出了t-SNE（ t-Distributed Stochastic Neighbor Embedding），通过构建对称的 SNE 解决因为不对称导致梯度计算复杂的情况；通过在低维嵌入空间中引入 t-分布 替换高斯分布，使用更加偏重长尾分布的方式将距离转化为概率分布，可以减轻不同类别的簇“拥挤”在一起，不容易区分的问题。

* 高维输入空样本点两两间距离概率计算

假设有N个高维对象x1​,…,xN​，t-SNE 首先计算与 xi​和xj​两两之间相似度（距离）成正比的概率pij​，对于i≠j，有，且pi∣i​=0，注意到对所有i，有∑j​pj∣i​=1。

数据点xj​到数据点xi​的距离是pj∣i​条件概率，即xi​选择xj​为其邻近点，当邻近点的选择与它们之间以xi​为高斯分布中心的概率密度成比例。定义高维输入空间中的联合概率pij​为对称条件概率，有​​。同样有， pii​=0，∑ij​pij​=1。

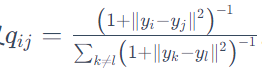
对应到 [van der Maaten 提供的代码](https://lvdmaaten.github.io/tsne/)，其高维输入空间样本点间距离计算对应代码为：

(n, d) = X.shape  
sum\_X = np.sum(np.square(X), 1)  
D = np.add(np.add(-2 \* np.dot(X, X.T), sum\_X).T, sum\_X)

高维输入空间样本点间距离高斯分布概率计算对应代码为：

def Hbeta(D=np.array([]), beta=1.0):  
 """  
 Compute the perplexity and the P-row for a specific value of the  
 precision of a Gaussian distribution.  
 """  
  
 # Compute P-row and corresponding perplexity  
 P = np.exp(-D.copy() \* beta)  
 sumP = sum(P)  
 H = np.log(sumP) + beta \* np.sum(D \* P) / sumP  
 P = P / sumP  
 return H, P  
  
# Compute P-values  
P = x2p(X, 1e-5, args['perplexity'])  
P = P + np.transpose(P)  
P = P / np.sum(P)  
P = P \* 4. # early exaggeration  
P = np.maximum(P, 1e-12)

* 低维嵌入空间随机初始化样本点两两间距离概率计算

t-SNE 的目标是将高维输入空间对象映射到低维嵌入空间，得到  
y1​,…,xN​， yi​∈Rd， d通常为2或3维。与Y两两样本点之间距离成比例的概率尽可能的接近高维输入空间的pij​，因此在低维嵌入空间使用类似的方法计算，只是将高斯分布替换为t-分布，从而能够保留更多具有较远距离的相似度。定义。

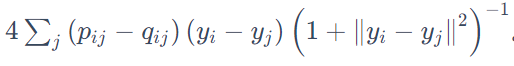
低维嵌入空样本点（随机初始化）距离和概率计算对应代码为：

Y = np.random.randn(n, no\_dims)  
# Compute pairwise affinities  
sum\_Y = np.sum(np.square(Y), 1)  
num = -2. \* np.dot(Y, Y.T)  
num = 1. / (1. + np.add(np.add(num, sum\_Y).T, sum\_Y))  
num[range(n), range(n)] = 0.  
Q = num / np.sum(num)  
Q = np.maximum(Q, 1e-12)

* 高低维度空间样本点距离概率相似性计算和优化迭代

测度高维输入空间和低维嵌入空间数据点两两间距离概率相似性，用 Kullback-Leibler 散度计算，公式为：

根据上述目标函数，使用梯度下降法，寻找低维嵌入空间数据点的向量，梯度为：

。

梯度计算对应代码为：

# Compute gradient  
PQ = P - Q  
for i in range(n):  
 dY[i, :] = np.sum(np.tile(PQ[:, i] \* num[:, i], (no\_dims, 1)).T \* (Y[i, :] - Y), 0)

从提供的示范代码中也可以看到，t-SNE 使用了预降维的技巧，先将高维数据降至低维，例如示范中将 784 维降至 50 维后再用 t-SNE 方法降维至指定的 2 维，达到优化的目的；同时，示范代码中对 概率乘以了一个倍数，例如 4，以避免因为概率过小导致的优化太慢的问题。t-SNE 主要用于高维数据的可视化，也可以从可视结果中估计到数据可能的分类数，及关联特征间的关系。

t-SNE 伪代码为：

**Algorithm 1**Simple version of t-Distributed Stochastic Neighbor Embedding.

**Data**: data set *X*={*x*1​,*x*2​,…,*xn*​},

cost function parameters: perplexity Perp,

optimization parameters: number of iterations *T*, learning rate *η*, momentum*α*(*t*).

**Result**: low-dimensional data representation Y(*T*)={*y*1​,*y*2​,…,*yn*​}.

**procedure**t-SNE()

compute pairwise afﬁnities *pj*∣*i*​ with perplexity Perp (using Equation 1)

set *pij*​=2*npj*∣*i*​+*pi*∣*j*​​

sample initial solution Y(0)={*y*1​,*y*2​,…,*yn*​} from N(0,10−4*I*)

**for***t*=1 to *T***do**

compute low-dimensional afﬁnities *qij*​ (using Equation 4)

compute gradient *δyδC*​ (using Equation 5)

set Y(*t*)=Y(*t*−1)+*ηδ*Y*δC*​+*α*(*t*)(Y(*t*−1)−Y(*t*−2))

**end for**

**end procedure**

使用 t-SNE 作者提供的示例数据和代码，计算如下。

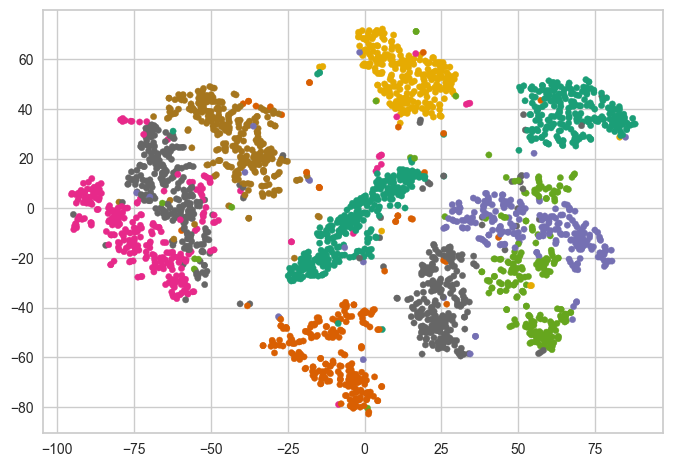
X=np.loadtxt(r"C:\Users\richie\omen\_richiebao\omen\_code\tsne\_python\mnist2500\_X.txt")  
labels=np.loadtxt(r"C:\Users\richie\omen\_richiebao\omen\_code\tsne\_python\mnist2500\_labels.txt")  
print(X.shape)

(2500, 784)

Y=usda\_manifold.tsne(X,2,max\_iter=1000,print\_interval=200)

Preprocessing the data using PCA...  
Computing pairwise distances...  
Mean value of sigma: 2.5733192000 of 2500...  
Iteration 1000: error is 0.979305

pylab.scatter(Y[:, 0], Y[:, 1], 20, labels,cmap='Dark2')  
pylab.show()

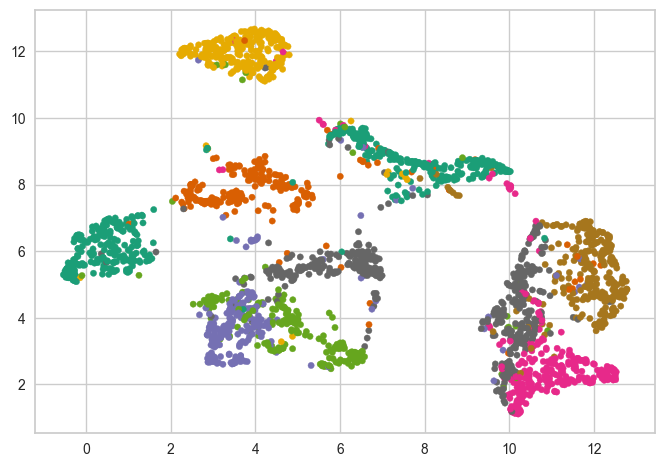


类似 t-SNE，McInnes, L.等人[20]提出了UMAP（Uniform Manifold Approximation and Projection）流形学习方法，该算法基于黎曼几何（Riemannian geometry）和代数拓扑（algebraic topology）的理论框架构建，在可视化质量上可与 t-SNE 相媲美，并且可以保留更多的全局结构，同时具有更好的运行性能。UMAP 的实现可以使用[UMAP](https://umap-learn.readthedocs.io/en/latest/index.html)⑦库。

用 UMAP 方法重复试验上述 t-SNE 输入数据，可以观察到 UMAP 表现出相对更好的团聚结果。

reducer=umap.UMAP(n\_components=2)  
embedding=reducer.fit\_transform(X)

pylab.scatter(embedding[:, 0], embedding[:, 1], 20, labels,cmap='Dark2')  
pylab.show()



### 3.7.6.2 街道空间多维景观指数的 UMAP 空间特征区段划分

再次使用街道空间多维景观指数数据，用 UMAP 方法结合 DBSCAN 聚类寻找多维景观指数在街道空间中分布的特征区段。

提取待降维的数据字段。

fields=['Green view index','Sky view factor', 'Ground view index','Equilibrium degree', 'Perimeter area ratio(mn)','Shape index(mn)', 'Fractal dimension(mn)', 'Color richness index','Key point size(0-10]', 'Key point size(10-20]','Key point size(30-40]', 'Key point size(20-30]','frank\_e', 'num']  
print(f'fields num = {len(fields)}')  
metrics\_selection\_df=metrics\_inall[fields]  
X=metrics\_selection\_df.to\_numpy()

fields num = 14

使用 UMAP 降维至3维。

reducer=umap.UMAP(n\_components=3)  
embedding=reducer.fit\_transform(X)  
  
metrics\_inall\_copy=metrics\_inall.copy(deep=True)  
metrics\_inall\_copy[['embedding\_x','embedding\_y','embedding\_z']]=embedding

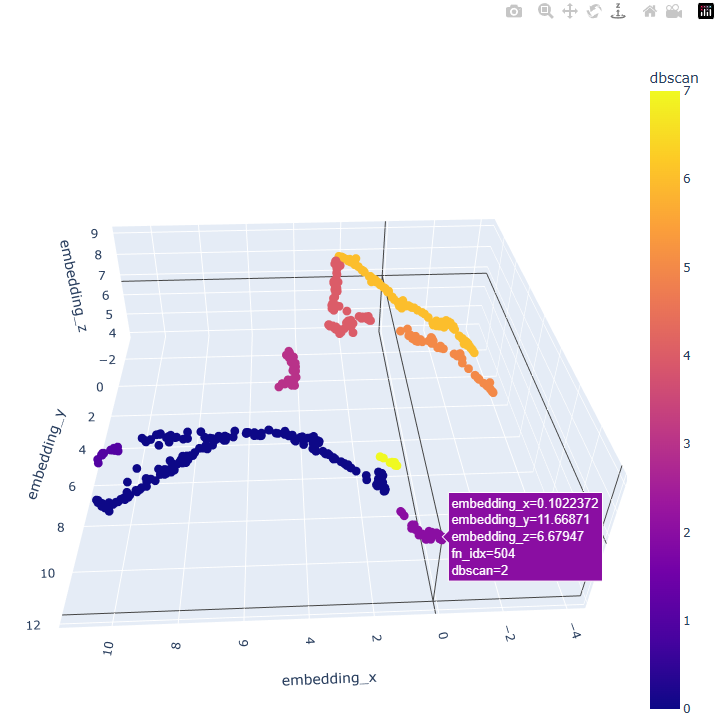
用 DBSCAN 聚类降维结果数据，通过调整参数eps，控制聚类簇数量。

clustering = DBSCAN(eps=0.6, min\_samples=3).fit(embedding)  
print(np.unique(clustering.labels\_))  
metrics\_inall\_copy['dbscan']=clustering.labels\_

[0 1 2 3 4 5 6 7]

按降维向量打印三维点，用聚类降维结果表示为点的颜色，观察数据点的分布情况。

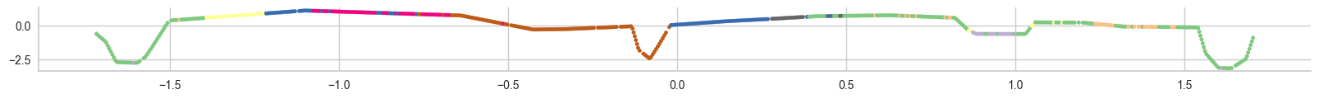
fig\_3d=px.scatter\_3d(  
 metrics\_inall\_copy, x='embedding\_x',y='embedding\_y',z='embedding\_z',   
 color='dbscan',  
 width=800, height=800,  
 hover\_data=['fn\_idx'],  
 )  
fig\_3d.update\_traces(marker\_size=5)  
fig\_3d.show()



为了观察实际街道空间下聚类降维后的结果，按照实际经纬度坐标的相对位置打印点并赋予聚类的颜色，得知多维景观指数的综合特征将街道划分成了多个区段，可以将该种方法拓展到类似的城市空间特征模式结构寻找问题上。

metrics\_inall\_copy[['lon','lat']]=metrics\_inall.apply(lambda row:[row.geometry.x,row.geometry.y],axis=1,result\_type='expand')  
norm\_lon=usda\_utils.normalize\_by\_meanNstd(metrics\_inall\_copy[['lon']])[0]  
norm\_lat=usda\_utils.normalize\_by\_meanNstd(metrics\_inall\_copy[['lat']])[0]

fig,ax=plt.subplots(figsize=(20,1))  
  
color=usda\_vis.vals4color\_cmap(metrics\_inall\_copy['dbscan'].values,cmap='Accent',vmin=metrics\_inall\_copy['dbscan'].min(), vmax=metrics\_inall\_copy['dbscan'].max())  
ax.scatter(norm\_lon,norm\_lat,s=10,color=color)   
   
ax.spines['top'].set\_visible(False)  
ax.spines['right'].set\_visible(False)  
plt.show()



注释（Notes）：

① Manifold learning on handwritten digits: Locally Linear Embedding, Isomap…，（<https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py>）。

② class sklearn.manifold.MDS，（<https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html>）。

③ sklearn.manifold.smacof，（<https://scikit-learn.org/stable/modules/generated/sklearn.manifold.smacof.html>）。

④ （sklearn）MDS 代码，（<https://github.com/scikit-learn/scikit-learn/blob/7f9bad99d/sklearn/manifold/_mds.py#L372>）。

⑤ Isotonic Regression 示例，（<https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_isotonic_regression.html>）。

⑥ Visualizing the stock market structure，（<https://scikit-learn.org/stable/auto_examples/applications/plot_stock_market.html#sphx-glr-auto-examples-applications-plot-stock-market-py>）。

⑦ UMAP Python 库，（<https://umap-learn.readthedocs.io/en/latest/index.html>）。

参考文献（References）:

[1] 周志华 著. 机器学习, 北京: 清华大学出版社, 2016年1月.(ISBN 978-7-302-206853-6)

[2] Lawrence, N. D. (2012). A Unifying Probabilistic Perspective for Spectral Dimensionality Reduction: Insights and New Models. Journal of Machine Learning Research, 13(51), 1609–1638. Retrieved from http://jmlr.org/papers/v13/lawrence12a.html

[3] Lee, John A.; Verleysen, Michel (2007). Nonlinear Dimensionality Reduction. Springer. ISBN 978-0-387-39350-6.

[4] Rajawat, K., & Kumar, S. (2016). Stochastic Multidimensional Scaling. arXiv [Math.OC]. Retrieved from http://arxiv.org/abs/1612.07089

[5] Kruskal,J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika, 29(1), 1–27. doi:10.1007/bf02289565

[6] Kruskal, J. B. (1964). Nonmetric multidimensional scaling: A numerical method. Psychometrika, 29(2), 115–129. doi:10.1007/bf02289694

[7] 同济大学数学系编,工程数学：线性代数（第5版）[M]. 高等教育出版社, 2007.05. ISBN： 9787040212181

[8] Burden, R. L., Faires, J. D., & Burden, A. M. (2015). Numerical Analysis. Retrieved from https://books.google.com/books?id=9DV-BAAAQBAJ

[9] Eigenvalues and eigenvectors\_Wikipedia, <https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors#cite_note-FOOTNOTEPressTeukolskyVetterlingFlannery200738-28>

[10] 李航.统计学习方法[M].北京,清华大学出版社.2012.03.ISBN: 9787302275954

[11] Schölkopf, B., Smola, A., & Müller, K.-R. (1997). Kernel principal component analysis. In W. Gerstner, A. Germond, M. Hasler, & J.-D. Nicoud (Eds.), Artificial Neural Networks — ICANN’97 (pp. 583–588). Berlin, Heidelberg: Springer Berlin Heidelberg.

[12] Raschka, S., & Mirjalili, V. (2019). Python Machine Learning: Machine Learning and Deep Learning with Python, Scikit-Learn, and TensorFlow 2, 3rd Edition. Retrieved from https://books.google.com/books?id=n1cjyAEACAAJ

[13] Tenenbaum, J. B., de Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. Science (New York, N.Y.), 290(5500), 2319–2323. doi:10.1126/science.290.5500.2319

[14] Lagrange multiplier\_Wikipedia, <https://en.wikipedia.org/wiki/Lagrange_multiplier>.

[15] Roweis, S. T., & Saul, L. K. (2000). Nonlinear Dimensionality Reduction by Locally Linear Embedding. Science, 290(5500), 2323–2326. doi:10.1126/science.290.5500.2323

[16] Ghojogh, B., Ghodsi, A., Karray, F., & Crowley, M. (2020). Locally Linear Embedding and its Variants: Tutorial and Survey. arXiv [Stat.ML]. Retrieved from http://arxiv.org/abs/2011.10925

[17] Sparse inverse covariance, <https://scikit-learn.org/stable/modules/covariance.html#sparse-inverse-covariance>

[18] Hinton, G. E., & Roweis, S. (2002). Stochastic Neighbor Embedding. In S. Becker, S. Thrun, & K. Obermayer (Eds.), Advances in Neural Information Processing Systems (Vol. 15). Retrieved from https://proceedings.neurips.cc/paper\_files/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf

[19] van der Maaten, L. J. P., & Hinton, G. E. (2008). Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research, 9(nov), 2579-2605.

[20] McInnes, L., Healy, J., & Melville, J. (2020). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv [Stat.ML]. Retrieved from http://arxiv.org/abs/1802.03426