同济大学
TONGJI UNIVERSITY

王睿智

ruizhiwang@tongji.edu.cn

人工智能技术与应用

无监督学习示例：客户分组

假设有许多篇新闻报道，其类别未知。若根据报道内容的相似性，将这些报道划分成几个组，使得每个组内的报道内容很相似，而不同组间的却很不同。

某购物网站已积累了许多客户的购买记录。如何利用这些购买记录和客户在网站的浏览活动，来理解这些顾客和他们所需？

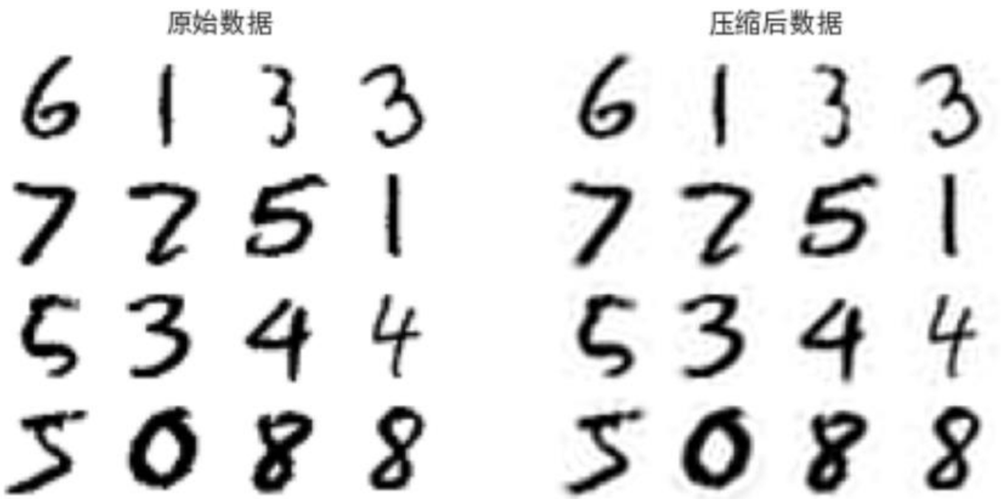
将这些客户分成几个组，确保每个组内的购买和浏览活动很相似，而不同组间的却很不同。客户分组可用于推荐系统，为在同组的其他客户推荐一些用户购买的商品。

与监督学习不同，这类学习无监督信号，仅根据数据本身进行学习，故称为无监督学习。

无监督学习示例：数据压缩

手写数字数据集（MNIST_784），每个样本是一个手写数字的28×28灰度图像，共有784个特征。维数很高，后续处理困难。

如何解决呢？压缩数据，降低特征的维数。



784个特征　　154个特征

- 不操作监督信号。
- 探索分析未标记数据中的内在模式。

4.1 降维

4.1.1 投影

4.1.2 流行学习

4.2 聚类

常见无监督学习任务

- 无监督学习只处理“特征”(X)，不操作监督信号（y），探索分析未标记数据中的内在模式。
- **常见无监督学习任务**：

多重共线性是指至少两个变量相关联的情况。

  - **降维**：模型中变量越多，多重共线性和模型过拟合可能性越高。此外，大量的特征会增加模型的复杂性以及调参和拟合时间。减少数据集中的特征数量，即降维，将有助于消除多重共线性、提高泛化性能和训练速度。此外，还可用于数据可视化、数据压缩等。
  - **聚类**：将高度相似的样本聚成组，使得组内高度同质性、组间高度异质性。可用于客户分组、搜索引擎、图像分割、半监督学习等。
  - **异常检测**：学习正常数据，以便能从中检测出异常个例。可用于生产线上的次品检测等。
  - **密度估计**：估计产生数据的随机过程的概率密度函数，密度估计结果常用于异常检测、数据分析、可视化。

- 当数据的维数很高时，将出现样本稀疏、距离计算困难等问题，机器学习问题变得相当困难，这种现象被称为“**维数灾难**”。
- 应对维数灾难的有效手段：

（1）**降维**(dimension reduction)

- 是缓解维数灾难的一个重要途径。通过某种数学变换，将原高维特征空间$\mathbb{R}^n$转变为一个低维“子空间”$\mathbb{R}^l\,(l < n)$，将x压缩在一个较小的表示中，同时损失的信息尽可能少。
- 主要途径两种：**投影**和**流形学习**。
- 降维用于机器学习的数据处理，还可用于数据可视化、数据压缩等。

（2）**分布式表示**：深度神经网络，以分布式表示带来指数增益，可有效解决维数灾难的挑战。

投影：通过坐标变换，将数据从原高维空间投影到新的低维空间。

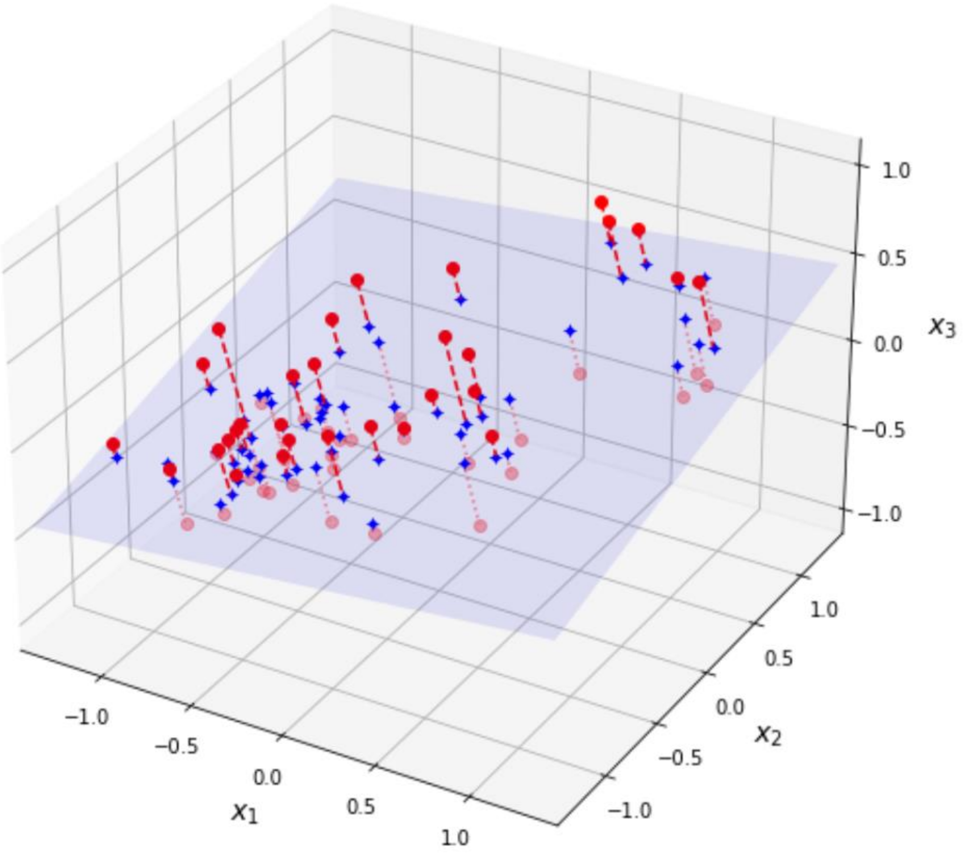
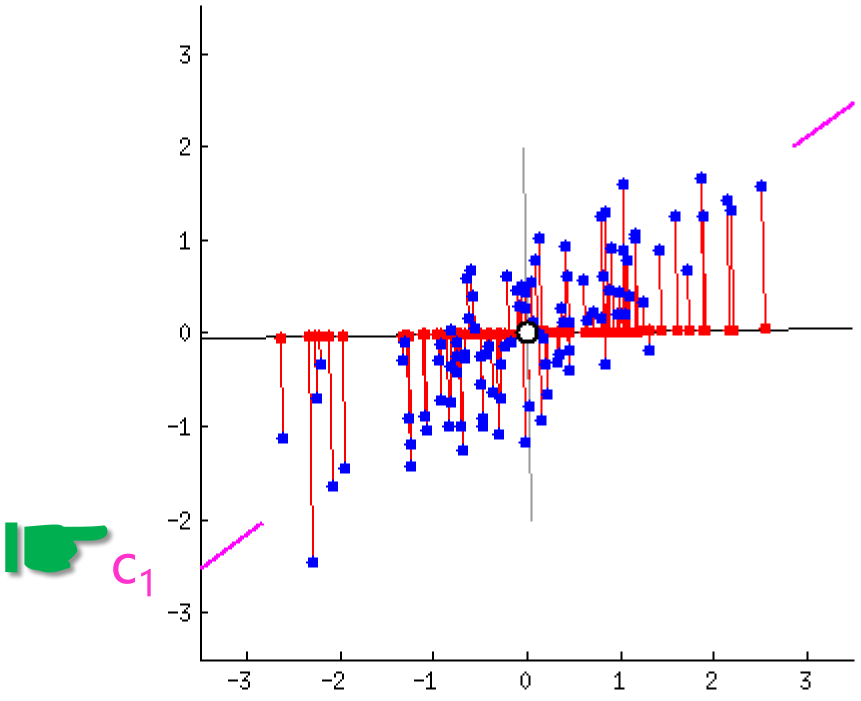


图1 一个分布靠近2D子空间（淡紫色超平面）的3D数据集

每个样本都向淡紫色超平面垂直地投影。维数降低了！！！ 且损失信息少！！

怎样投影损失信息最少?

4.1.1 投影



如图中的2D数据，通过投影方式降维为1D，投影到哪条线上最佳？为什么？

- c_1最佳。向c_1投影将保持原数据最多的信息。
- c_1是数据差异最大的方向，即方差最大的方向。
- 数据差异越大的方向，保持原数据中的信息也就越多。
- 降维时，向方差大的方向投影 → **PCA**

主成分分析 (principal component analysis，简称 PCA)是最流行的投影降维算法之一。

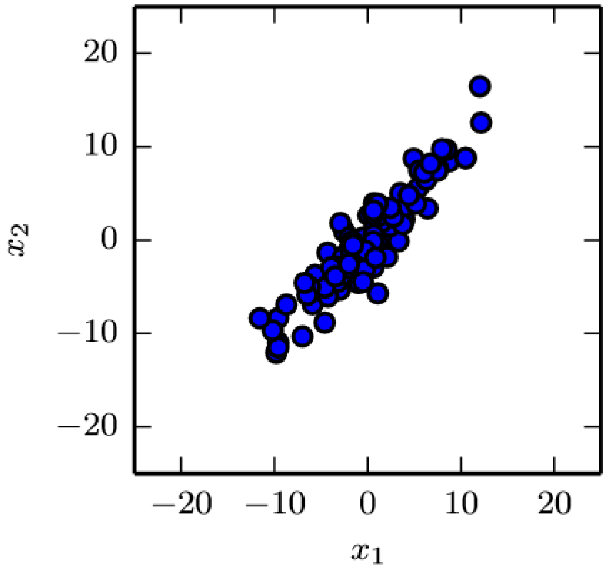
PCA 学习一种线性投影，旨在找到高维数据中方差最大的方向，作为新的坐标轴，将数据投影到新的坐标系（坐标轴相互正交）。

新坐标系的第1个坐标轴，选择原始数据中方差最大的方向；第2个坐标轴选择和第1个坐标轴正交且具有次大方差的方向，… 。

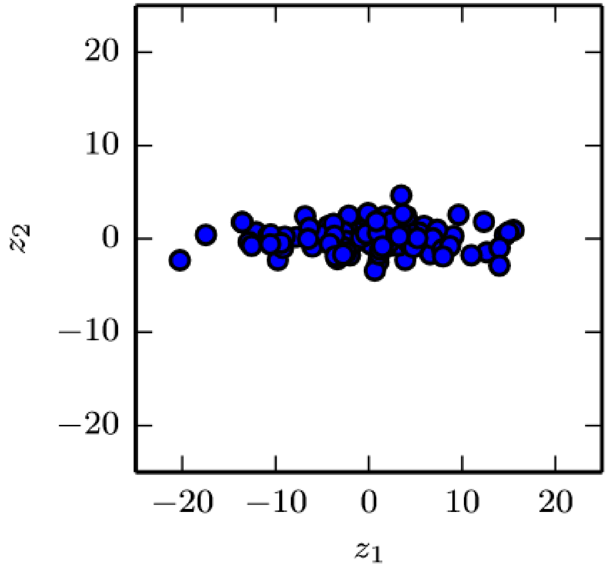
该过程重复进行，重复次数为原始特征的数目或指定降到的维数。

新坐标系的第i轴被称为数据的**第 i 主成分**，可由训练数据集（注意数据的均值为零）的奇异值分解（Singular Value Decompositon, 简称SVD）得到。

一个重要性质：PCA具有将原始数据变换为元素之间彼此不相关表示的能力。



原始数据



变换后的数据

PCA通过线性投影（$\boldsymbol{W}$是投影矩阵），将数据集最大方差的方向和新空间的轴对齐。

(左) 原始数据$\boldsymbol{x} = [x_1, x_2]$，在这个空间中，数据方差的方向与轴的方向不是对齐的。

(右) 变换过的数据$\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W}$在轴$z_1$的方向上有最大的变化，第二大变化方差的方向沿着$z_2$。

PCA算法的主要步骤：

输入：数据集 $\boldsymbol{X}=(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)},\cdots,\boldsymbol{x}^{(m)})^T$，$d$维空间，$m$个样本；
降维后空间的维数为k。

PCA对特征取值范围不同敏感

过程：

1. 标准化数据集，得$\boldsymbol{X}_{std}$；
2. 构造$\boldsymbol{X}_{std}$的协方差矩阵；
3. 计算协方差矩阵的特征值和特征向量；
4. 取最大的k个特征值所对应的特征向量$\boldsymbol{w}_1,\boldsymbol{w}_2,\cdots,\boldsymbol{w}_k$，构成投影矩阵$\boldsymbol{W}$。
5. 用投影矩阵$\boldsymbol{W}$对d维输入数据集$\boldsymbol{X}$进行变换，获得新的k维特征子空间投影 $\boldsymbol{X}_{pca}=\boldsymbol{X}\cdot\boldsymbol{W}$。

1：标准化数据集，得$\boldsymbol{X}_{std}$
2：对$\boldsymbol{X}_{std}$做SVD分解
3：取前k个右奇异向量构成投影矩阵$\boldsymbol{W}$

输出：投影矩阵$\boldsymbol{W}=(\boldsymbol{w}_1,\boldsymbol{w}_2,\cdots,\boldsymbol{w}_k)$，降维后的数据集$\boldsymbol{X}_{pca}$。

例4.1 定义并测试PCA函数

（1）自定义pca(X,topN)函数，将X投影到topN维的子空间中，返回投影变换后的矩阵，以及其逆变换到原空间的结果。

（2）生成一个有200个数据点的二维数据集。调用自定义pca函数对该数据集降维。可视化降维结果。

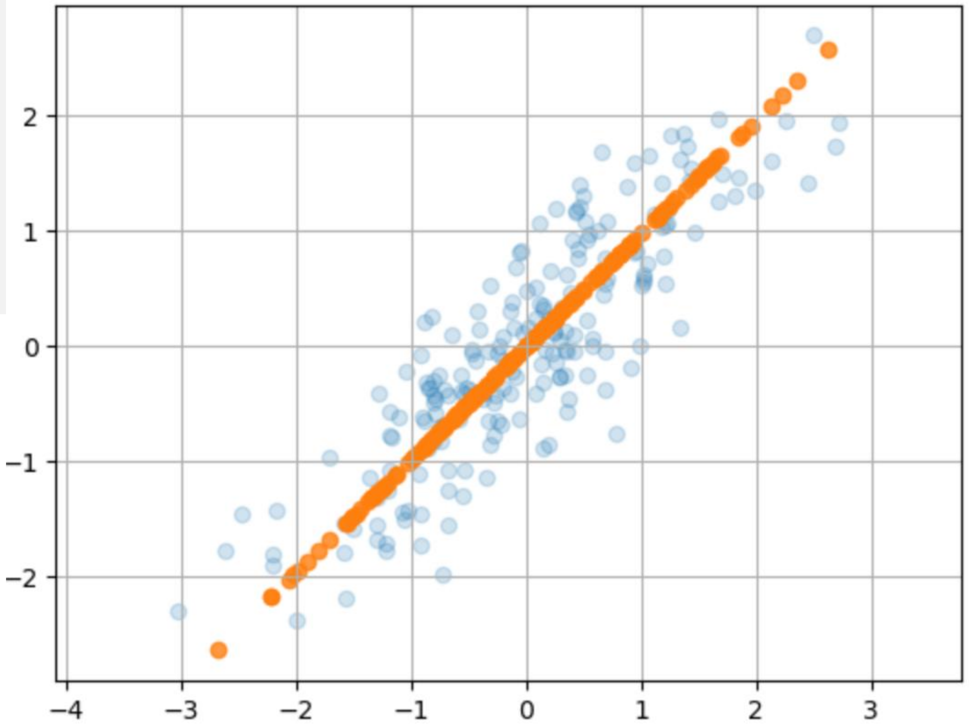
1.自定义PCA函数

输入：原始数据X(每列表示一个特征；行是样本), 低维空间维度topN；
输出：降维后数据X_pca，降维后重构数据，X_recon ，即逆变换到原始空间

```
import numpy as np
from sklearn.preprocessing import StandardScaler
def pca(X,topN):
    """参数：原始数据X（数组），低维空间维度topN（整数）"""
    sc = StandardScaler()
    X_std = sc.fit_transform(X)   # 1.标准化数据
   U,S,Vt = np.linalg.svd(X_std) # 2.调SVD()分解函数，返回U,s，VT：X_std的右奇异矩阵的转置
    W = Vt.T[:,:topN]              # 3.取前topN个右奇异向量构成投影矩阵
    X_pca = X_std.dot(W)           # 4.将原始数据转换到新空间，得到降维后数据
    X_recon = sc.inverse_transform(X_pca * W.T) # 5.降维后数据重构，用于与原始数据比较
    return X_pca,X_recon
```

2.调用自定义PCA降维并可视化

```
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.RandomState(0)
X = np.dot(rng.randn(200,2),rng.rand(2,2))   # 200个数据点的二维数据
lowX,reconX = pca(X,1)                       # 调用自定义pca函数
plt.scatter(X[:,0],X[:,1],alpha=0.2)         # alpha控制透明度，介于0(透明)和1(不透明)之间
reconX = np.array(reconX)
plt.scatter(reconX[:,0],reconX[:,1],alpha=0.8)
plt.axis('equal')
plt.grid()
```



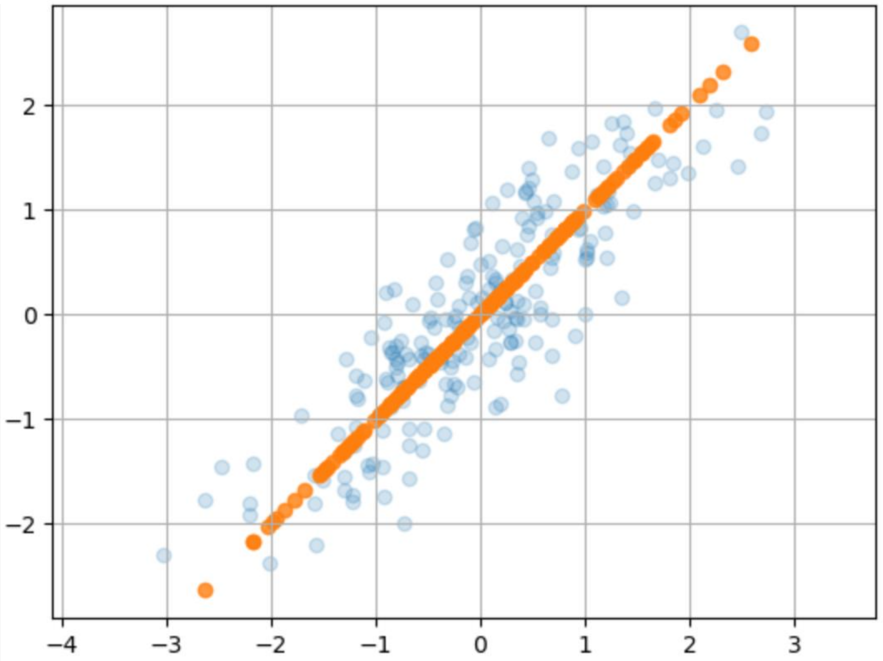
Sklearn.decomposition中提供了PCA

PCA(*n_components=None*, *svd_solver='auto'*, *random_state=None,…*)

参数	• ***n_components*** : int,float,字符串，默认值None。要保留的成分数目，若未设置，则保留所有成分n_components == min(n_samples, n_features)。 • ***svd_solver*** **:** {'auto', 'full', 'arpack', 'randomized'}, default='auto'
方法	• **fit(X)** 使用X拟合模型。 • **fit_transform(X[, y])** 使用X拟合模型并在X上应用降维 • **inverse_transform(X)** 将数据转换回原始空间 • **transform(X)** 将降维应用于X.
属性	• **n_components_** 整型，若n_componets取值[0,1],则为从数据中估计的成分数；否则与n_components值相同。 • **explained_variance_ratio_** 数组型, 形状 (*n_components*,) 所有主成分对原数据集方差的解释比率。

例4.1（续）调用sklearn中PCA

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

rng = np.random.RandomState(3)
X = np.dot(rng.randn(200,2),rng.rand(2,2))  # 200个数据点
X_std = StandardScaler().fit_transform(X)   # 标准化
pca = PCA(n_components=1)
X_pca=pca.fit_transform(X_std) # 降维后
# 为演示降维的效果，将降维的数据逆变换，并与原始数据一起画出来
X_new=pca.inverse_transform(X_pca)
plt.scatter(X[:,0],X[:,1],alpha=0.2)
plt.scatter(X_new[:,0],X_new[:,1],alpha=0.8)
plt.axis('equal')
plt.grid()
print("第一主成分占方差比:",*pca.explained_variance_ratio_)
```



第一主成分贡献了数据集93%的方差

第一主成分占方差比: 0.9297718585301104

例4.2 红酒数据可视化

（1）自定义pca(X,topN)函数，对红酒数据降维处理，只保留第一、二主成分，画出散点图。

（2）调用sklearn的PCA，也降到2维，画出散点图。

（3）在降维后的数据集上，训练逻辑回归分类器，并测试其结果。显示第一、二主成分占方差比。

1.读入数据

```
import pandas as pd
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)
df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue',
                   'OD280/OD315 of diluted wines', 'Proline']
```

```
df_wine.head()
```

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

2.数据划分、标准化

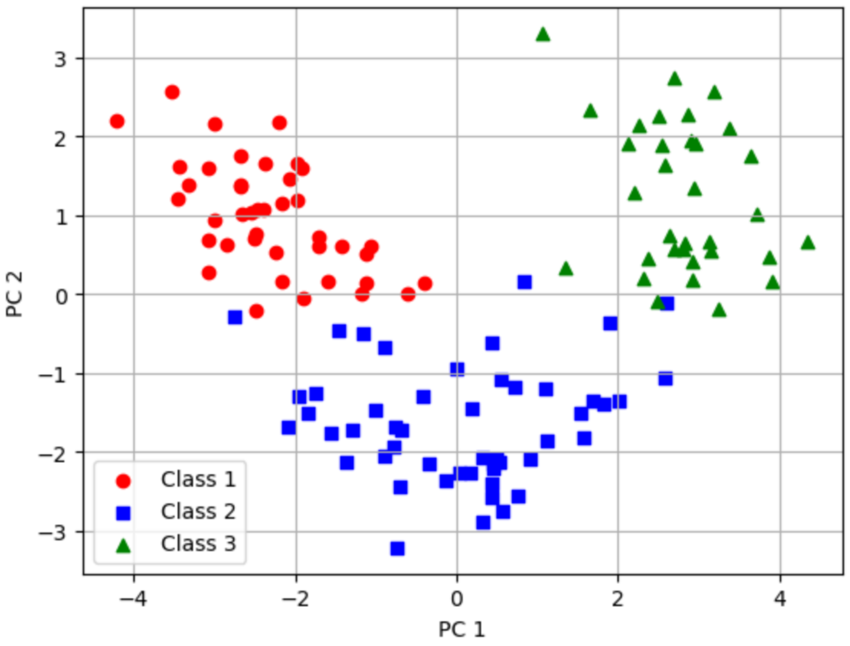
```
from sklearn.model_selection import train_test_split
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    stratify=y,
                                                    random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

3.利用SVD分解求前2个主成分

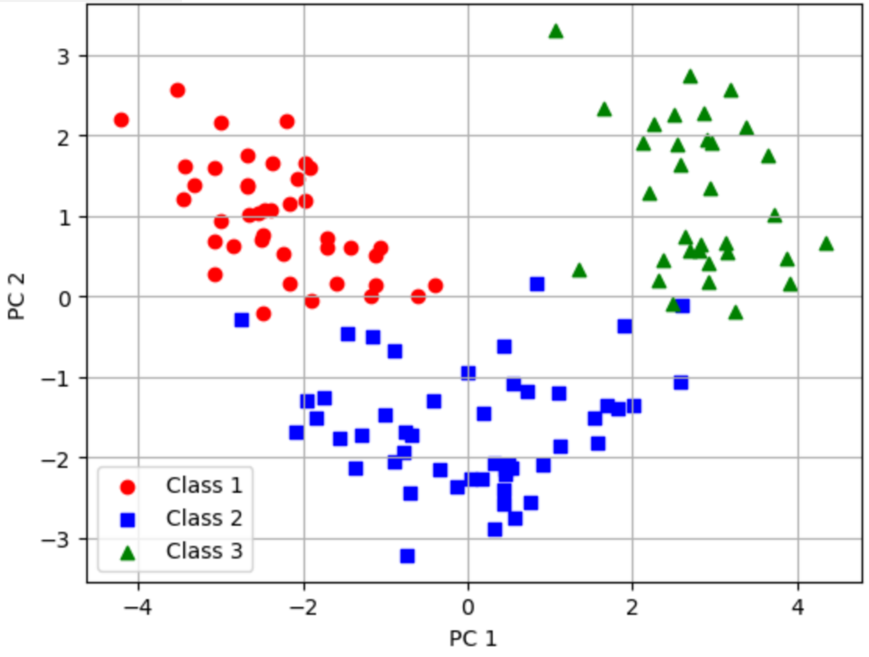
```
import numpy as np
U,S,Vt = np.linalg.svd(X_train_std) # 2.调SVD()分解函数，返回U,s,VT
W = Vt.T[:,:2]
```

```
import matplotlib.pyplot as plt
X_train_pca = X_train_std.dot(W)
colors = ['r', 'b', 'g']
markers = ['o', 's', '^']
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train == l, 0],
                X_train_pca[y_train == l, 1],
                c=c, label=f'Class {l}', marker=m)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.grid()
```



4.调用sklearn的PCA

```
from sklearn.decomposition import PCA
pca2=PCA(n_components=2)
pca2.fit(X_train_std)
X_train_pca=pca2.transform(X_train_std) #降维后
colors = ['r', 'b', 'g']
markers = ['o', 's', '^']
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train == l, 0],
                X_train_pca[y_train == l, 1],
                c=c, label=f'Class {l}', marker=m)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.grid()
```



5. 用二维数据训练逻辑回归分类，并测试

```
from sklearn.linear_model import LogisticRegression

X_test_pca = pca2.transform(X_test_std)

lr = LogisticRegression(random_state=1)
lr = lr.fit(X_train_pca, y_train)
print(f'训练得分{lr.score(X_train_pca,y_train):.4f}')
print(f'测试得分{lr.score(X_test_pca,y_test):.4f}')
print("第1、2主成分占方差比:",*pca2.explained_variance_ratio_)
```

```
训练得分0.9839
测试得分0.9259
第1、2主成分占方差比: 0.3695146859960769 0.1843492705988409
```

若提升其分类性能，应确定合适的主成分数。

确定合适的成分数量

- 实际使用PCA降维时，正确估计用于描述数据的成分数量(即维数)是非常关键的环节。
- 常用方法：

① 选择维数d使得累计方差贡献率足够大(如 95%), 然后设置n_components = d重新运行PCA。

```
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum>=0.95) +1
pca = PCA(n_components = d)
X_reduced = pca.fit_transform(X_train)
```

cumsum>=0.95 结果是布尔数组
(False, False,⋯,**True**,True,..,True)

numpy.**argmax**(*a*)
返回数组a最大值的索引

② 直接设置n_components 为0.0到1.0之间的float，如 0.95。

```
pca = PCA(n_components = 0.95)
X_reduced = pca.fit_transform(X_train)
```

③ 将累计方差贡献率看作是关于成分数量的函数，画图找拐点，从而确定所需成分的数量。

④ **与后续监督学习方法相结合，找出使得分类(或回归)得分最高的成分数量。**

手写数字数据集(MNIST_784)训练数据集有60000个样本，784个特征。①使用PCA降维该数据，保留95%方差。② 画出维数与explained_variance_radio_的关系图。③用随机森林分类器，随机搜索出分类得分最高的成分数。

1. 导入数据

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns;sns.set()
import pandas as pd

df_train = pd.read_csv('mnist_train.csv',header=None)
X_train = np.array(df_train.iloc[:,1:].values)
print(f"训练集X_train:{X_train.shape}")
```

训练集X_train:(60000, 784)

2. PCA压缩数据

保留95%方差

```
from sklearn.decomposition import PCA
pca=PCA(n_components=0.95)  # 保留95%的方差，即累计方差贡献率95%
X_reduced=pca.fit_transform(X_train)
print(f'成分数量：{pca.n_components_}')
print(f'这些成分的累计方差贡献率:{pca.explained_variance_ratio_.sum()}')
```

成分数量：154
这些成分的累计方差贡献率：0.9501960192613034

3. 压缩效果可视化

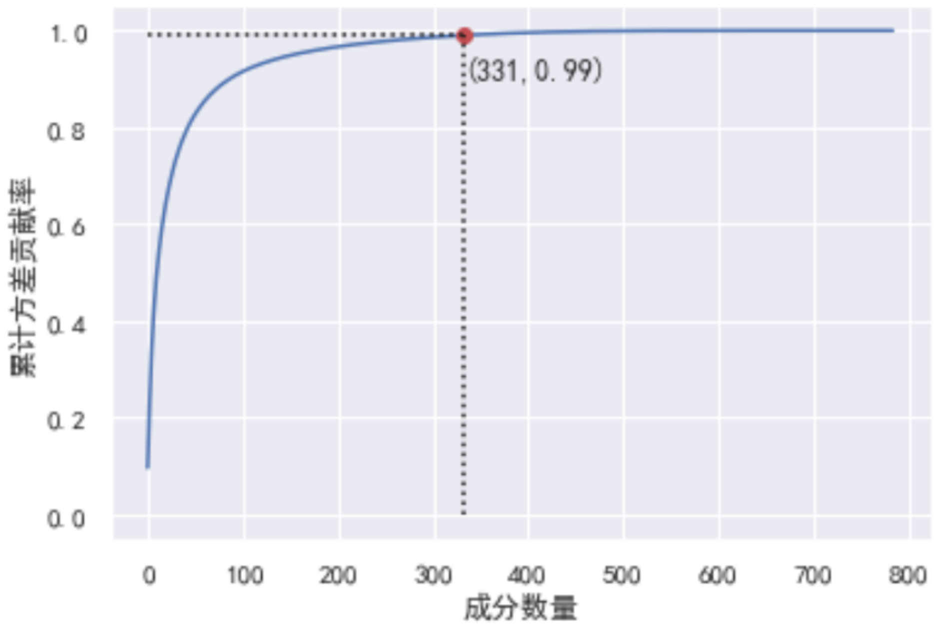
```
X_recovered = pca.inverse_transform(X_reduced) # 将数据转换回原始空间

# 下面两句是防止jupyter notebook绘图中文显示乱码
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

plt.figure(figsize=(8, 4))
for idx, X in enumerate((X_train[::2000], X_recovered[::2000])):
    plt.subplot(1, 2, idx + 1)
    plt.title(["原始数据", "压缩后数据"][idx])
    for row in range(4):
        for col in range(4):
            plt.imshow(X[row * 4 + col].reshape(28, 28), cmap="binary",
                       vmin=0, vmax=255, extent=(row, row + 1, col, col + 1))
            plt.axis([0, 4, 0, 4])
            plt.axis("off")
```

4. 画出“累计方差贡献率”与“成分数量”的关系图

```
pca=PCA().fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
dims = np.argmax(cumsum >= 0.99) + 1  # 保留99%的方差，需要多少个成分？
print(f'若保留99%方差，至少需要{dims}个成分')
```

若保留99%方差，至少需要331个成分

```
plt.plot(cumsum)
plt.xlabel('成分数量')
plt.ylabel('累计方差贡献率')
plt.plot(dims, 0.99, "ro")
plt.text(dims,0.9,'(%d,0.99)'%dims)
plt.plot([dims, dims], [0, 0.99], "k:")
plt.plot([0, dims], [0.99, 0.99], "k:")
```



从上面的量化曲线可以看出在前N个主成分中包含了多少“总784维”的方差。如，前100个成分包含了几乎86%的方差。若要包含近99%的方差，就需要331个成分。

5. 采用随机森林分类器，随机搜索出分类得分最高的成分数

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import make_pipeline

clf = make_pipeline(PCA(random_state=42),
                    RandomForestClassifier(random_state=42))
param_distrib = {"pca__n_components": np.arange(10, 180),
                 "randomforestclassifier__n_estimators": np.arange(50, 500)}
rnd_search = RandomizedSearchCV(clf, param_distrib, n_iter=10, cv=3,
                                random_state=42)
X_train = np.array(df_train.iloc[:30000,1:].values)
y_train = np.array(df_train.iloc[:30000,0].values)
rnd_search.fit(X_train, y_train)
```

```
print(f'搜索结果: {rnd_search.best_params_}')
```

```
搜索结果: {'randomforestclassifier__n_estimators': 323, 'pca__n_components': 45}
```

当处理速度要求高，或者内存受限，或者数据是非线性的（如瑞士卷），如何解决?

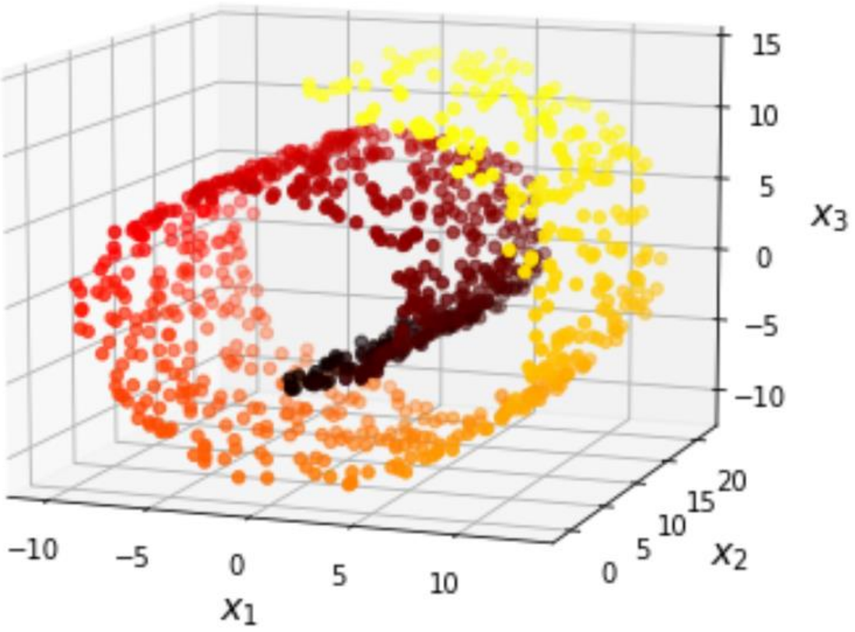
- **随机PCA算法** 随机选出主成分，速度快
- **增量PCA算法** 小批处理，解决内存受限问题

from sklearn.decomposition import IncrementalPCA

- **核PCA算法** 核技巧，实现复杂的非线性投影

from sklearn.decomposition import KernelPCA 非线性降维

4.1.2 流形学习



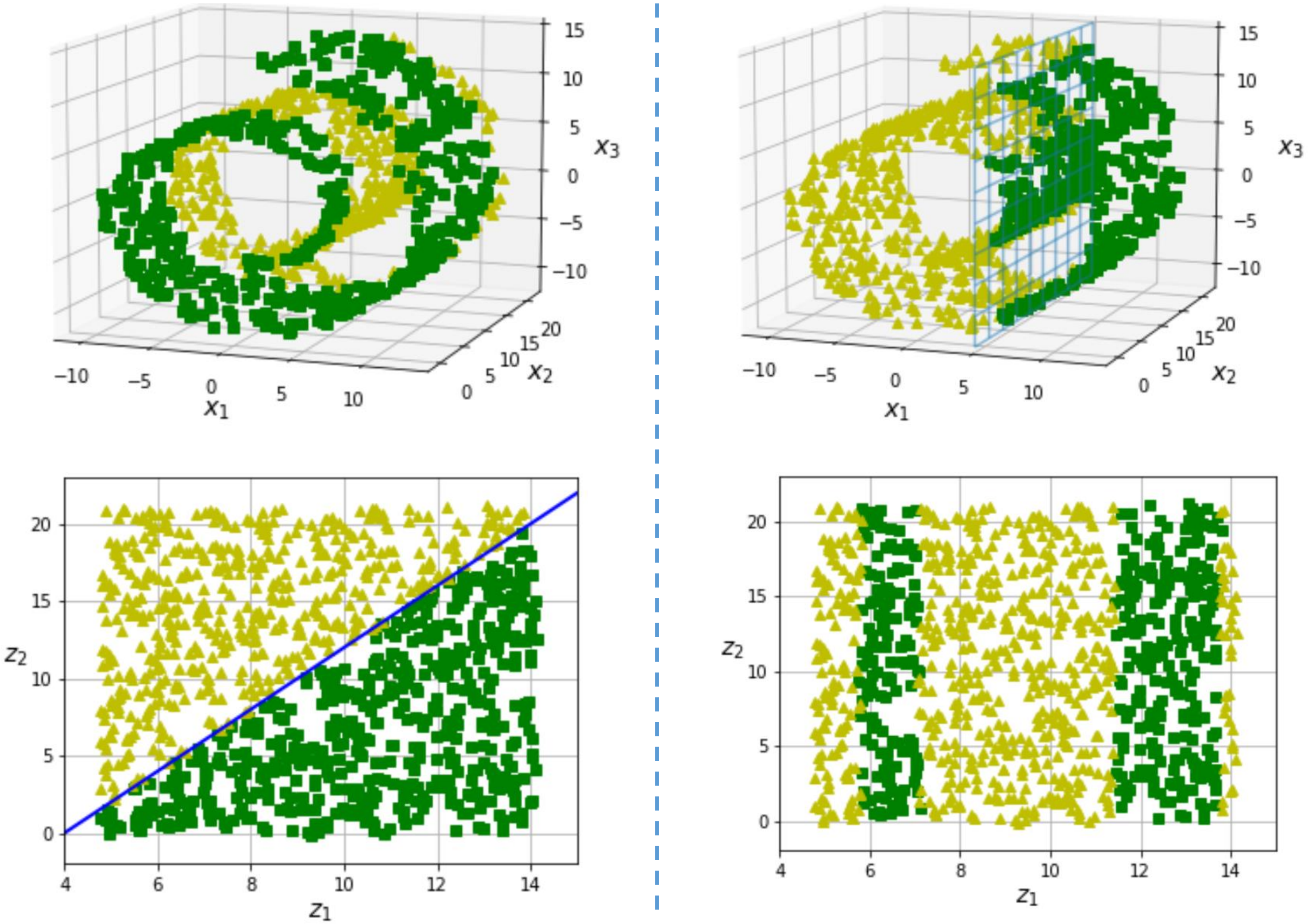
大多数真实世界中的高维数据集都接近于低维流形(manifold)。—— **流形假设**

左图，瑞士卷是一个2维流形，一个在3维空间中被卷曲的2维形状。

流形是指嵌在高维空间中的低维拓扑空间。一个d维流形是一个n维空间(d<n)中的一部分，它局部地类似于一个d维的超平面，在n维空间中被弯曲、扭曲。

许多降维算法通过对训练数据所在的流形建模来工作，这称为**流形学习（manifold learning）**。

流形假设还伴随另一个**隐式假设**：低维流形对于后续的分类或回归任务更简单。



如图中，左侧数据集的低维流形有利于分类，而右侧则不利于分类。

流形学习对分类或回归不一定有帮助，但对探索数据的非线性特征有意义，**常用于数据可视化。**

Locally Linear Embedding(LLE)

- 是一个功能比较强的非线性降维算法。
- 可以有效地展开扭曲的流形，特别是当数据没有太多噪声时。
- **基本思想：**

首先，衡量每个训练样本与它最近邻的线性邻近程度。

然后，寻找能**保持**训练集的这些**局部线性关系**低维表示。

Sklearn.manifold中提供了Locally Linear Embedding类

```
LocallyLinearEmbedding(n_neighbors=5, n_components=2,
                       random_state=None,… )
```

参数	• ***n_neighbors*** : int，默认值5。每个样本要考虑的近邻数量。 • ***n_components*** : int，默认值2。流形的坐标数。
方法	• **fit(X)** 计算数据X的嵌入向量。 • **fit_transform(X[, y])** 计算数据X的嵌入向量，变换X并返回。

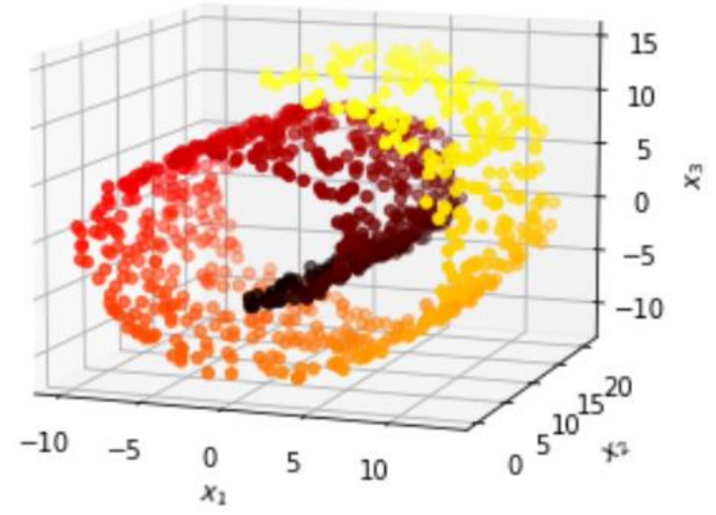
`make_swiss_roll(n_samples=100, noise=0.0, random_state=None)` 产生瑞士卷数据集

参数	• ***n_samples*** : int, 默认值100。数据集中样本数. • ***noise*** : float, 默认值0.0。高斯噪声的标准偏差。
返回值	• X : 形状为(n_samples, 3)的数组。样本点, **三维** • t : 形状为 (n_samples,)的数组。根据流形中点的主要维度，样本的单变量位置。

1. 生成瑞士卷数据并可视化

```
from sklearn.datasets import make_swiss_roll
X_swiss, t = make_swiss_roll(n_samples=1000, noise=0.2, random_state=42)
```

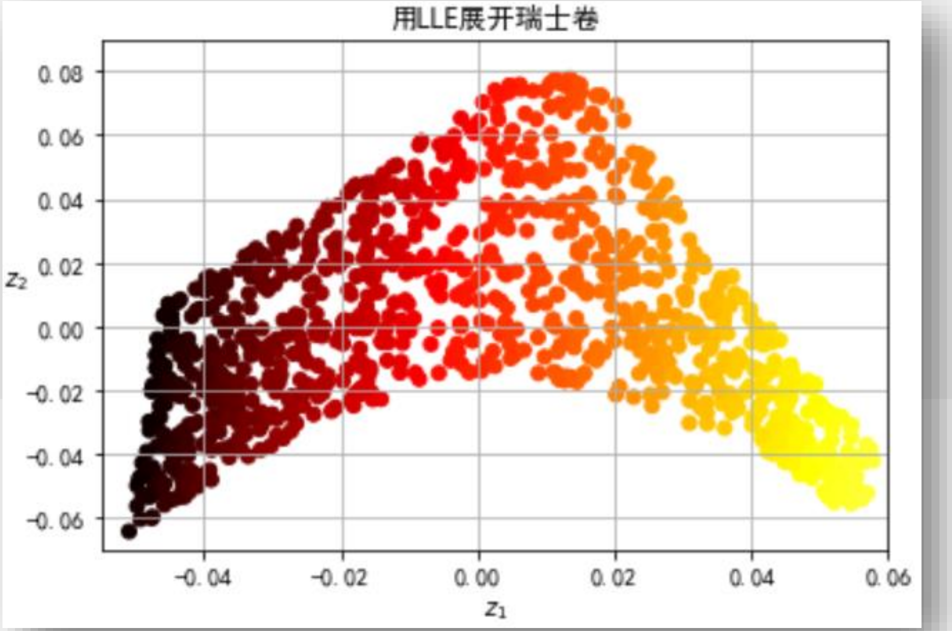
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
darker_hot = ListedColormap(plt.cm.hot(np.linspace(0, 0.8, 256)))
axes = [-11.5, 14, -2, 23, -12, 15]
fig = plt.figure(figsize=(6, 5))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_swiss[:, 0], X_swiss[:, 1], X_swiss[:, 2],
           c=t, cmap=darker_hot)
ax.view_init(10,-70) # 设置仰角为10度，方位角为-70度，
ax.set_xlabel("$x_1$", labelpad=6, rotation=0)
ax.set_ylabel("$x_2$", labelpad=6, rotation=0)
ax.set_zlabel("$x_3$", labelpad=6, rotation=0)
plt.show()
```



2. LLE

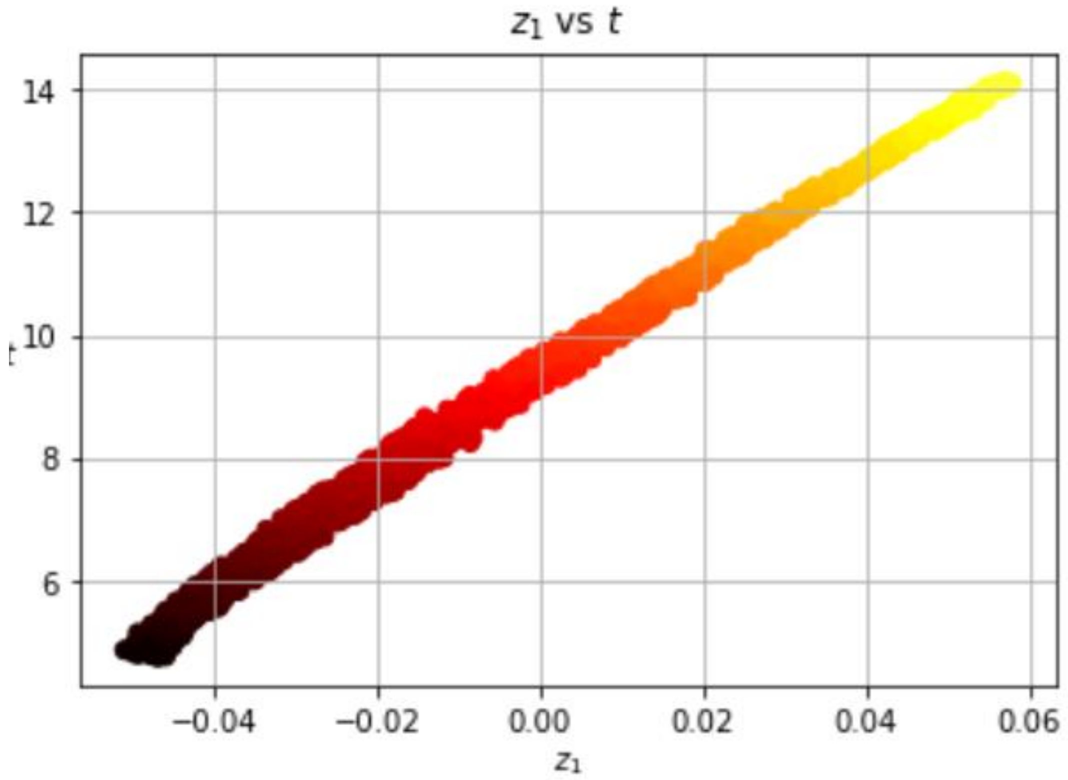
```
from sklearn.manifold import LocallyLinearEmbedding

lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10, random_state=42)
X_unrolled = lle.fit_transform(X_swiss)
```

```
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

plt.scatter(X_unrolled[:, 0], X_unrolled[:, 1], c=t, cmap=darker_hot)
plt.xlabel("$z_1$")
plt.ylabel("$z_2$", rotation=0)
plt.axis([-0.055, 0.060, -0.070, 0.090])
plt.grid(True)

plt.title("用LLE展开瑞士卷")
plt.show()
```



3. LLE 效果：显示z1与t的相关性

```
plt.title("$z_1$ vs $t$")
plt.scatter(X_unrolled[:, 0], t, c=t, cmap=darker_hot)
plt.xlabel("$z_1$")
plt.ylabel("$t$", rotation=0)
plt.grid(True)
plt.show()
```



小结

- **PCA**：应用广泛、可解释性强。
  - ✓ 降维
  - ✓ 数据可视化
  - ✓ 噪声过滤
  - ✓ 数据压缩
- PCA变体：RandomizedPCA 、IncrementalPCA、KernelPCA

- **LLE**：流形学习算法，主要用于非线性降维

作业九 9.1 PCA可视化iris数据

（1）利用PCA对iris降维到二维空间

（2）可视化降维后的数据，画出散点图

（3）通过逻辑回归对变换后的样本进行分类，并测试其性能

作业九

9.2 给定下列的2维数据集X，怎样用PCA进行降维?

(1)计算投影矩阵W

(2)求出投影后数据集

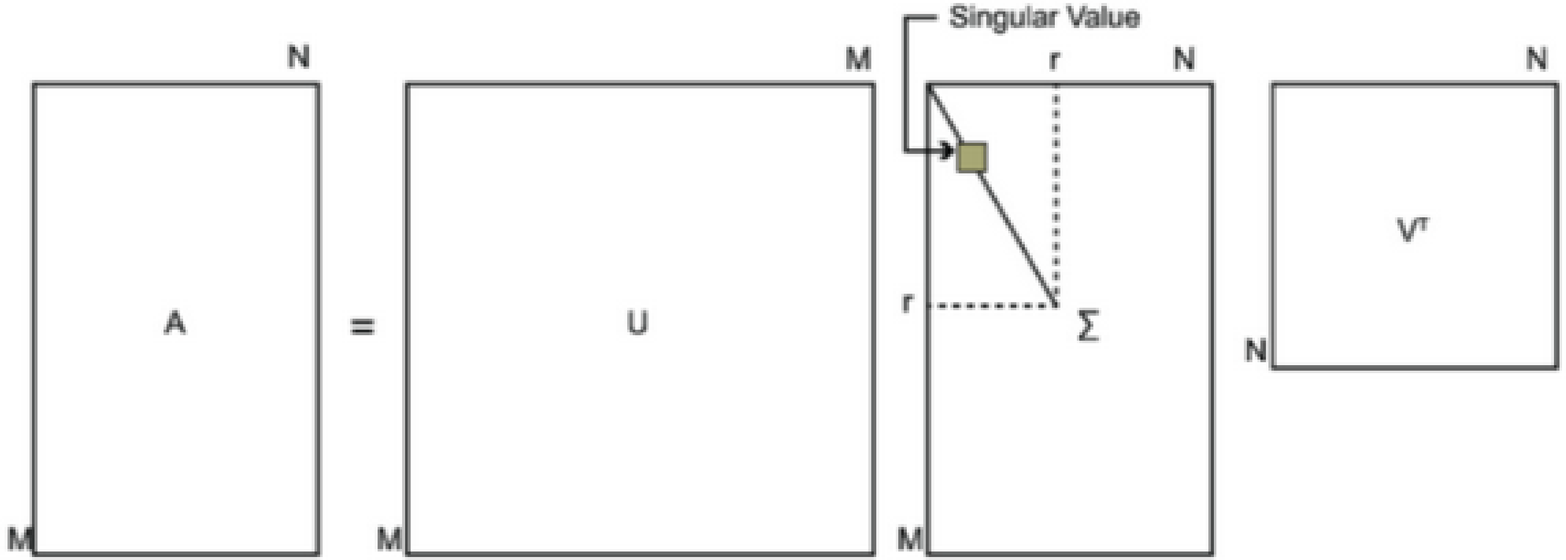
$$
\begin{array}{cc}
 & \begin{array}{cc} x_1 & x_2 \end{array} \\
D = & \begin{bmatrix} 2.5 & 2.4 \\ 0.5 & 0.7 \\ 2.2 & 2.9 \\ 1.9 & 2.2 \\ 3.1 & 3.0 \\ 2.3 & 2.7 \\ 2.0 & 1.6 \\ 1.0 & 1.1 \\ 1.5 & 1.6 \\ 1.1 & 0.9 \end{bmatrix}
\end{array}
$$

矩阵的奇异值分解

将矩阵A（是$m \times n$的矩阵）分解为三个矩阵乘积：$A = U\Sigma V^T$

U是$m \times m$的矩阵(m是样本数目)；V是$n \times n$的矩阵（n是特征数目）。

Σ 是$m \times n$的对角阵，除了主对角线上的元素以外全为0，主对角线上的每个元素都称为奇异值；U和V都是酉矩阵，即满足$U^T U = I, V^T V = I$



如何求出SVD分解后的U,Σ,V这三个矩阵呢？ $A=U\Sigma V^T$

如果将$\boldsymbol{A}$的转置和$\boldsymbol{A}$做矩阵乘法，那么会得到n×n的一个方阵$\boldsymbol{A}^T\boldsymbol{A}$，可以进行特征分解，得到的特征值和特征向量满足下式：

$$(\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{v}_i=\lambda_i\boldsymbol{v}_i$$

这样就得到矩阵$\boldsymbol{A}^T\boldsymbol{A}$ 的n个特征值和对应的n个特征向量$\boldsymbol{v}$。将$\boldsymbol{A}^T\boldsymbol{A}$ 的所有特征向量张成一个n×n的矩阵$\boldsymbol{V}$，就是SVD公式里的$\boldsymbol{V}$矩阵。一般将$\boldsymbol{V}$中的每个特征向量称为$\boldsymbol{A}$的右奇异向量。

如果将$\boldsymbol{A}$和$\boldsymbol{A}^T$做矩阵乘法，那么会得到m×m的一个方阵$\boldsymbol{A}\boldsymbol{A}^T$,**进行特征值分解，得到**矩阵$\boldsymbol{A}\boldsymbol{A}^T$的m个特征值和对应的m个特征向量$\boldsymbol{u}$。将$\boldsymbol{A}\boldsymbol{A}^T$的所有特征向量张成一个m×m的矩阵$\boldsymbol{U}$，就是SVD公式里面的$\boldsymbol{U}$矩阵。一般将$\boldsymbol{U}$中的每个特征向量叫做$\boldsymbol{A}$的左奇异向量。

$\boldsymbol{\Sigma}$ 对角线上的元素称为矩阵$\boldsymbol{A}$的奇异值，**A**的非零奇异值是$\boldsymbol{A}^T\boldsymbol{A}$的特征值的平方根，也是$\boldsymbol{A}\boldsymbol{A}^T$的特征值的平方根。

每个右奇异向量对应一个主成分。若将数据从n维降到d维，可通过线性变换 $\boldsymbol{X}_{d-proj}=\boldsymbol{X}\boldsymbol{V}_d$ 实现，其中 $\boldsymbol{V}_d$是由$\boldsymbol{V}$的前d列（即前d个主成分）。