第7讲 PCA降维

- PCA相关概念
- PCA算法步骤
- sklearn.decomposition.PCA

PCA相关概念

- **数据降维**：采用某种映射方法，将原高维空间中的数据点映射到低维度空间中

  - 数据降维的本质是学习一个映射函数

    f : x -> y

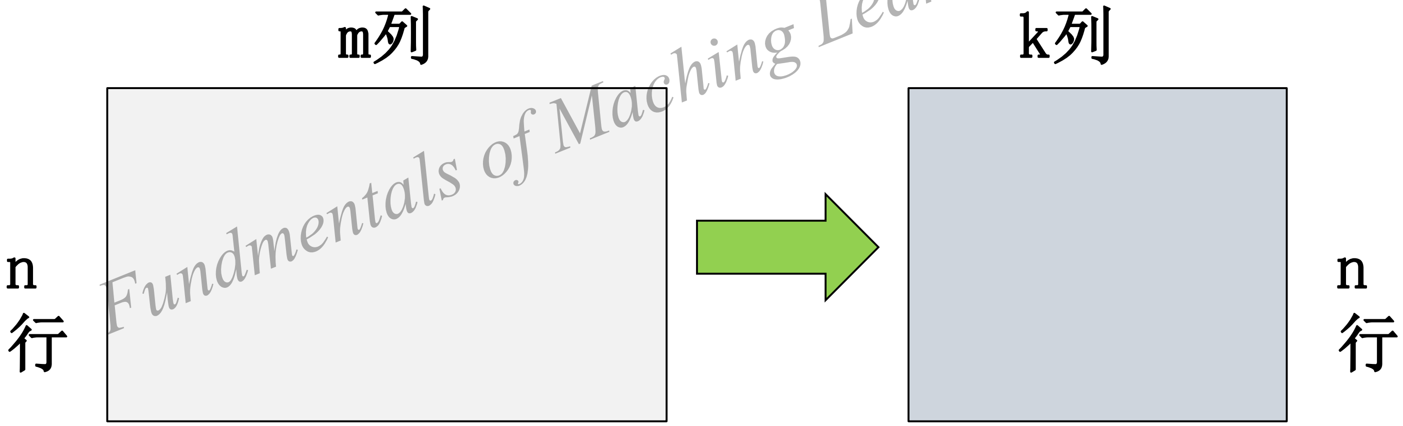
    x: 原始数据点的表达，目前最多使用向量形式

    y: 数据点映射后的低维向量，通常其维度远远小于x的维度

    f: 显式或隐式、线性或非线性

PCA相关概念

- **PCA (Principal Component Analysis)**：将具有相关性的高维变量综合成线性无关的低维变量，这些变量称为主成分（principal component）



- 主成分能可尽可能保留原始数据的信息

PCA相关概念

- PCA思想：将 m 维特征映射到 l 维空间($m \gg l$)，去除原始特征之间的冗余性（通过去除相关性手段达到）

原始数据向方差最大的方向进行投影。一旦发现方差最大的投影方向，则继续寻找次方差大的方向进行投影，……

PCA相关概念

- **方差(variance)**：统计中的方差（样本方差）是各个数据分别与其平均数之差的平方和的平均数。

假设有n个数据，记为$X=\{x_i\}$ $(i=1,\dots,n)$

$$Var(X)=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu)^2 \qquad u=\frac{1}{n}\sum_{i=1}^{n}x_i$$

方差描述了样本数据的波动程度

PCA相关概念

- **协方差(covariance)**：度量两个随机变量关系的统计量

假设有n个两维变量数据，记为$(X,Y)=\{(x_i,y_i)\}\ (i=1,\dots,n)$

$$\mathrm{cov}(X,Y)=\frac{1}{n}\sum_{i=1}^{n}(x_i-E(X))(y_i-E(Y))$$

其中$E(X)$和$E(Y)$分别是X和Y的样本均值，分别定义如下

$$E(X)=\frac{1}{n}\sum_{i=1}^{n}x_i,\qquad E(Y)=\frac{1}{n}\sum_{i=1}^{n}y_i$$

- 衡量两个变量之间的相关度

PCA相关概念

- 协方差衡量两个变量之间的相关度
  - 当协方差cov(X, Y) > 0 时，称X与Y正相关
  - 当协方差cov(X, Y) < 0 时，称X与Y负相关
  - 当协方差cov(X, Y) = 0 时，称X与Y线性不相关

PCA相关概念

- 皮尔逊相关系数(Pearson Correlation coefficient)：
  将两组变量之间的关联度规整到一定的取值范围内

$$\mathrm{corr}(X,Y)=\frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}=\frac{Cov(X,Y)}{\sigma_x\sigma_y}$$

皮尔逊相关系数的性质：

- corr(X, Y) = corr(Y, X)
- |corr(X, Y)| <= 1
- |corr(X, Y)| =1的充要条件是存在常数b和c, 使得y = bx + c

PCA相关概念

相关性(correlation)与独立性(independence)

- 如果X和Y线性不相关，则 |**corr(X, Y)**| = **0**
- 如果X和Y彼此独立，则一定 |**corr(X, Y)**| = **0**，即X和Y不存在任何线性或非线性关系

PCA相关概念

- **协方差矩阵（Covariance matrix）：**
  由多个随机变量中两两变量的协方差组成的矩阵
- **示例，三维变量的协方差矩阵:**

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 1 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} \operatorname{cov}(x_1, x_1) & \operatorname{cov}(x_1, x_2) & \operatorname{cov}(x_1, x_3) \\ \operatorname{cov}(x_2, x_1) & \operatorname{cov}(x_2, x_2) & \operatorname{cov}(x_2, x_3) \\ \operatorname{cov}(x_3, x_1) & \operatorname{cov}(x_3, x_2) & \operatorname{cov}(x_3, x_3) \end{bmatrix}$$

- 对角线上的元素为方差：
  cov(x1, x1)=var(x1), cov(x2,x2)=var(x2), cov(x3,x3)=var(x3)

PCA相关概念

- **示例：计算X的协方差矩阵**

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 1 \end{bmatrix} = [x_1 \ x_2 \ x_3]$$

① 求每个维度的平均值

$$\bar{X} = [2. \ 1.5 \ 2.] = [\bar{x}_1 \ \bar{x}_2 \ \bar{x}_3]$$

② X的每个元素中心化

$$X = X - \bar{X} = \begin{bmatrix} -1 & 0.5 & 1 \\ 1 & -0.5 & -1 \end{bmatrix}$$

③ 计算协方差矩阵

$$\Sigma = \frac{1}{n-1} X^T X = \begin{bmatrix} 2 & -1 & -2 \\ -1 & 0.5 & 1 \\ -2 & 1 & 2 \end{bmatrix}$$

PCA相关概念

- 示例：计算X的协方差矩阵

```python
import numpy as np

X = np.array([(1, 2, 3), (3, 1, 1)])

X = X-np.mean(X, axis=0)

X_cov = np.cov(X, rowvar=False)
```

```python
print(X_cov)
```

```
[[ 2.  -1.  -2. ]
 [-1.   0.5  1. ]
 [-2.   1.   2. ]]
```

NumPy

特征值和特征向量

- 如果一个向量v是方阵A的特征向量，则一定可表示成

$$Av = \lambda v$$

λ被称为特征向量v对应的特征值

- 特征值分解：方阵A矩阵分解成

$$A = Q \Sigma Q^{-1}$$

Q 是矩阵A的特征向量组成的矩阵，

Σ 是一个对角阵，每一个对角线上的元素是一个特征值

PCA相关概念

◆ 示例：求特征值与特征向量

```
import numpy as np
X = [[-1, 1, 0], [-4, 3, 0], [1, 0, 2]]
eigenvalue,featurevector = np.linalg.eig(X)
print("原始矩阵的特征值" , X)
print("eigenvalue=", eigenvalue)
print("featurevector=\n", featurevector)
```

```
原始矩阵的特征值 [[-1, 1, 0], [-4, 3, 0], [1, 0, 2]]
eigenvalue= [2. 1. 1.]
featurevector=
 [[ 0.          0.40824829  0.40824829]
 [ 0.          0.81649658  0.81649658]
 [ 1.         -0.40824829 -0.40824829]]
```

PCA算法步骤

Method 1：特征值分解的PCA算法

输入： n个d维样本数据所构成的矩阵$\mathbf{X}$，降维后的维数l

输出：降维后的数据矩阵$\mathbf{Y}$

处理：

1: 将$\mathbf{X}$的每一列零均值化，即减去每一行的均值

2: 计算原始样本数据$\mathbf{X}$的协方差矩阵：$\boldsymbol{\Sigma} = \frac{1}{n-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{X}$

3: 求协方差矩阵$\boldsymbol{\Sigma}$的特征值及特征向量

4: 特征值按其值大到小排序，取前l个最大特征值所对的应特征向量$\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_l$ 组成映射矩阵$\mathbf{W}$

5: 计算降维后的数据矩阵：$\mathbf{Y}_{n\times l} = \mathbf{X}_{n\times d}\mathbf{W}_{d\times l}$

PCA算法步骤

◆ 示例：PCA降维

```
import numpy as np
data = np.array([(2.5, 2.4, 2.0), (0.5, 0.7, 2.1),
                 (2.2, 2.9, 1.0), (1.9, 2.2, 1.5)])

meanValues = np.mean(data, axis=0)
meanRemoved = data - meanValues
covMat = np.cov(meanRemoved, rowvar=False)

eigVal, eigVect = np.linalg.eig(covMat)
eigVal_sorted = np.argsort(-eigVal)
eigVal_sorted = eigVal_sorted[:2]
w = eigVect[:, eigVal_sorted]

pca_data = np.dot(meanRemoved, w)
```

```
print(pca_data)
```

```
[[-0.62845531  0.61282271]
 [ 1.90955221 -0.06431616]
 [-1.0536027  -0.46381884]
 [-0.2274942  -0.08468771]]
```

PCA算法步骤

如何选择PCA主成分数目？

- 根据累积贡献率的大小取前面m个(m<p)主成分
- 选取原则：

$$\frac{\sum_{i=1}^{m-1}\lambda_i}{\sum_{i=1}^{p}\lambda_i} < 80\sim85\% \qquad \text{并且} \qquad \frac{\sum_{i=1}^{m}\lambda_i}{\sum_{i=i}^{p}\lambda_i} \geq 80\sim85\%$$

PCA算法步骤

Method 2：奇异值分解PCA算法

输入：n个d 维样本数据所构成的矩阵$\mathbf{X}$，降维后的维数l

输出：降维后的数据矩阵$\mathbf{Y}$

处理：

1: 对矩阵X 进行截断奇异值分解，得到：$X = USV^T$，保留l奇异值、奇异向量

2: 奇异值按其值大到小排序，取前l个最大奇异值所对应的特征向量$\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_l$ 组成映射矩阵$\mathbf{W}$

3: 计算降维后的数据矩阵：$\mathbf{Y}_{n \times l} = \mathbf{X}_{n \times d} \mathbf{W}_{d \times l}$

sklearn.decomposition.PCA

- **class :** sklearn.decomposition.**PCA**(*n_components=None*, ***, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*)

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#

sklearn.decomposition.PCA

fit(self, X[, y])	Fit the model with X.
fit_transform(self, X[, y])	Fit the model with X and apply the dimensionality reduction on X.
get_covariance(self)	Compute data covariance with the generative model.
get_params(self[, deep])	Get parameters for this estimator.
get_precision(self)	Compute data precision matrix with the generative model.
inverse_transform(self, X)	Transform data back to its original space.
score(self, X[, y])	Return the average log-likelihood of all samples.
score_samples(self, X)	Return the log-likelihood of each sample.
set_params(self, **params)	Set the parameters of this estimator.
transform(self, X)	Apply dimensionality reduction to X.

sklearn.decomposition.PCA

◆ **示例：PCA降维**

```
import numpy as np

# 生成3维数据：4X3
data = np.array([(2.5, 2.4, 2.0), (0.5, 0.7, 2.1),
                 (2.2, 2.9, 1.0), (1.9, 2.2, 1.5)])
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
sklearn_pca_data = pca.fit_transform(data)

print("降维后的数据", sklearn_pca_data)
```

```
print("降维后的数据", sklearn_pca_data)
```

```
降维后的数据 [[-0.62845531  0.61282271]
 [ 1.90955221 -0.06431616]
 [-1.0536027  -0.46381884]
 [-0.2274942  -0.08468771]]
```

```
print("主成分贡献率", pca.explained_variance_ratio_)
print("主成分方差值", pca.explained_variance_)
```

```
主成分贡献率 [0.89594111 0.10365702]
主成分方差值 [1.73439266 0.20066272]
```

sklearn.decomposition.PCA

◆ 示例：对红酒数据集的样本特征降维

```python
# 导入红酒数据集
from sklearn.datasets import load_wine
wine = load_wine()
X = wine.data
y = wine.target

# 导入数据预处理工具
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print(X_scaled.shape)
```

```
(178, 13)
```

```python
# PCA降维用于数据可视化
from sklearn.decomposition import PCA

# 设置主成分数量为2（整数），浮点数表示按照“贡献率”选取主成分
pca = PCA(n_components=2)
pca.fit(X_scaled)
X_pca = pca.transform(X_scaled)

# 显示PCA后的数据维度
print(X_pca.shape)
```

```
(178, 2)
```

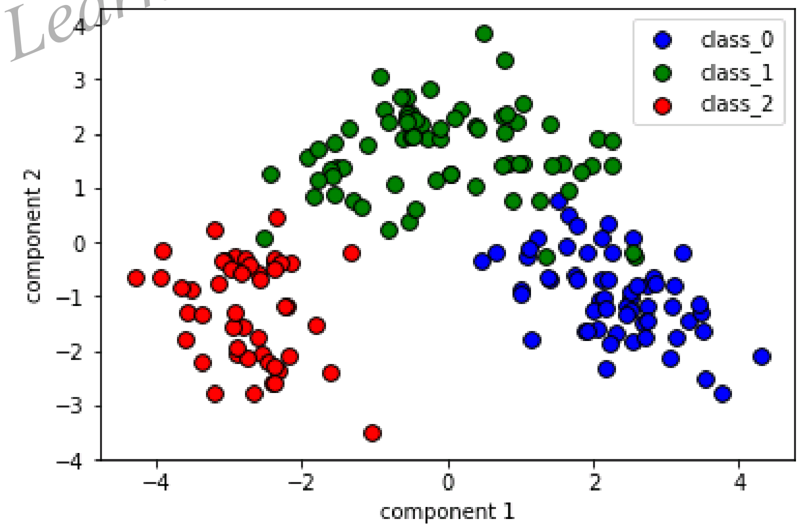
sklearn.decomposition.PCA

◆ 示例：对红酒数据集的样本特征降维

```
# PCA降维后数据的可视化
import matplotlib.pyplot as plt
%matplotlib inline
# 提取三个分类中的主成分
X0 = X_pca[wine.target==0]
X1 = X_pca[wine.target==1]
X2 = X_pca[wine.target==2]
# 绘制散点图
plt.scatter(X0[:,0],X0[:,1],c='b',s=60,edgecolor='k')
plt.scatter(X1[:,0],X1[:,1],c='g',s=60,edgecolor='k')
plt.scatter(X2[:,0],X2[:,1],c='r',s=60,edgecolor='k')

# 设置图注
plt.legend(wine.target_names, loc='best')
plt.xlabel('component 1')
plt.ylabel('component 2')
```

<matplotlib.text.Text at 0x12b17208>



sklearn.decomposition.PCA

- **示例：对红酒数据集的样本特征降维**

```
# 绘制主成分热度图
plt.matshow(pca.components_, cmap='plasma')
# 纵轴为主成分数
plt.yticks([0,1],['component 1','component 2'])
plt.colorbar()
# 横轴为原始特征数量
plt.xticks(range(len(wine.feature_names)),wine.feature_names, rotation=60,ha='left')
plt.show()
```