第16章 主成分分析

1.假设x为m 维随机变量，其均值为μ，协方差矩阵为Σ。

考虑由m维随机变量x到m维随机变量y的线性变换 $y_i=\alpha_i^T x=\sum_{k=1}^m \alpha_{ki} x_k, \quad i=1,2,\cdots,m$

其中$\alpha_i^T=(\alpha_{1i},\alpha_{2i},\cdots,\alpha_{mi})$。

如果该线性变换满足以下条件，则称之为总体主成分：

（1）$\alpha_i^T\alpha_i=1, i=1,2,\cdots,m$；

（2）$\operatorname{cov}(y_i,y_j)=0(i\neq j)$;

（3）变量y_1是x的所有线性变换中方差最大的；y_2是与y_1不相关的x的所有线性变换中方差最大的；一般地，y_i是与$y_1,y_2,\cdots,y_{i-1},(i=1,2,\cdots,m)$都不相关的$x$的所有线性变换中方差最大的；这时分别称$y_1,y_2,\cdots,y_m$为$x$的第一主成分、第二主成分、…、第$m$主成分。

2.假设x是m维随机变量，其协方差矩阵是Σ，Σ的特征值分别是$\lambda_1\geq\lambda_2\geq\cdots\geq\lambda_m\geq 0$，特征值对应的单位特征向量分别是$\alpha_1,\alpha_2,\cdots,\alpha_m$，则$x$的第2主成分可以写作

$y_i=\alpha_i^T x=\sum_{k=1}^m \alpha_{ki} x_k, \quad i=1,2,\cdots,m$ 并且，x的第i主成分的方差是协方差矩阵Σ的第i个特征值，即$\operatorname{var}(y_i)=\alpha_i^T\Sigma\alpha_i=\lambda_i$

3.主成分有以下性质：

主成分y的协方差矩阵是对角矩阵$\operatorname{cov}(y)=\Lambda=\operatorname{diag}(\lambda_1,\lambda_2,\cdots,\lambda_m)$

主成分y的方差之和等于随机变量x的方差之和 $\sum_{i=1}^m \lambda_i=\sum_{i=1}^m \sigma_{ii}$ 其中σ_{ii}是x_2的方差，即协方差矩阵Σ的对角线元素。

主成分y_k与变量x_2的相关系数$\rho(y_k,x_i)$称为因子负荷量（factor loading），它表示第k个主成分y_k与变量x的相关关系，即y_k对x的贡献程度。$\rho(y_k,x_i)=\frac{\sqrt{\lambda_k}\alpha_{ik}}{\sqrt{\sigma_{ii}}}, \quad k,i=1,2,\cdots,m$

4.样本主成分分析就是基于样本协方差矩阵的主成分分析。

给定样本矩阵 $X=\begin{bmatrix} x_1 & x_2 & \cdots & x_n\end{bmatrix}=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn}\end{bmatrix}$

其中$x_j=(x_{1j},x_{2j},\cdots,x_{mj})^T$是$x$的第$j$个独立观测样本，$j=1,2，\ldots,n$。

X的样本协方差矩阵 $\begin{gathered} S=[s_{ij}]_{m\times m}, \quad s_{ij}=\frac{1}{n-1}\sum_{k=1}^n(x_{ik}-\overline{x}_i)(x_{jk}-\overline{x}_j) \\ i=1,2,\cdots,m, \quad j=1,2,\cdots,m \end{gathered}$

给定样本数据矩阵X，考虑向量x到y的线性变换$y = A^T x$ 这里

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \end{bmatrix}$$

如果该线性变换满足以下条件，则称之为样本主成分。样本第一主成分$y_1 = a_1^T x$是在$a_1^T a_1 = 1$条件下，使得$a_1^T x_j (j = 1, 2, \cdots, n)$的样本方差$a_1^T S a_1$最大的$x$的线性变换；

样本第二主成分$y_2 = a_2^T x$是在$a_2^T a_2 = 1$和$a_2^T x_j$与$a_1^T x_j (j = 1, 2, \cdots, n)$的样本协方差$a_1^T S a_2 = 0$条件下，使得$a_2^T x_j (j = 1, 2, \cdots, n)$的样本方差$a_2^T S a_2$最大的$x$的线性变换；

一般地，样本第i主成分$y_i = a_i^T x$是在$a_i^T a_i = 1$和$a_i^T x_j$与$a_k^T x_j (k < i, j = 1, 2, \cdots, n)$的样本协方差$a_k^T S a_i = 0$条件下，使得$a_i^T x_j (j = 1, 2, \cdots, n)$的样本方差$a_k^T S a_i$最大的$x$的线性变换。

5.主成分分析方法主要有两种，可以通过相关矩阵的特征值分解或样本矩阵的奇异值分解进行。

（1）相关矩阵的特征值分解算法。针对$m \times n$样本矩阵X，求样本相关矩阵 $R = \frac{1}{n-1} X X^T$ 再求样本相关矩阵的k个特征值和对应的单位特征向量，构造正交矩阵 $V = (v_1, v_2, \cdots, v_k)$

V的每一列对应一个主成分，得到$k \times n$样本主成分矩阵 $Y = V^T X$

（2）矩阵X的奇异值分解算法。针对$m \times n$样本矩阵X $X' = \frac{1}{\sqrt{n-1}} X^T$ 对矩阵X'进行截断奇异值分解，保留k个奇异值、奇异向量，得到 $X' = U S V^T$ V的每一列对应一个主成分，得到$k \times n$样本主成分矩阵Y $Y = V^T X$

本章代码直接使用Coursera机器学习课程的第六个编程练习。

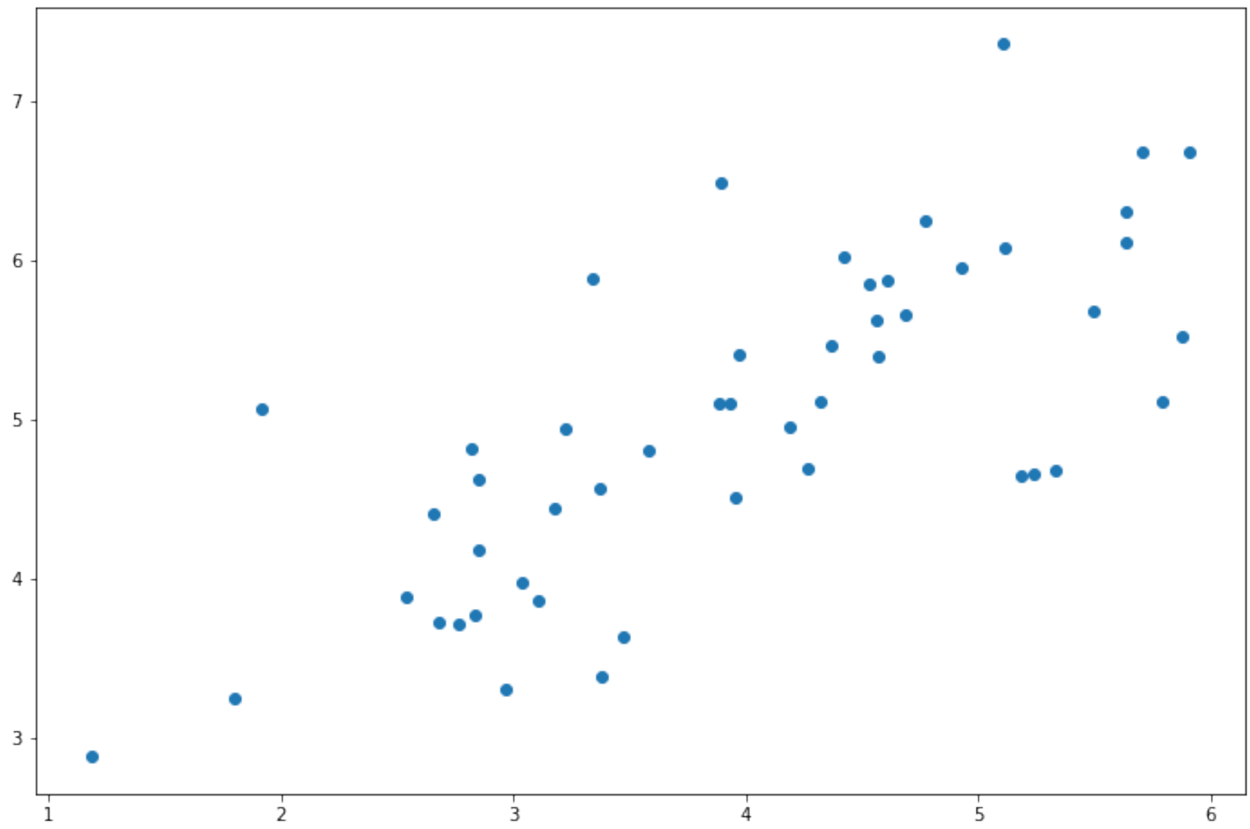
PCA（principal components analysis）即主成分分析技术旨在利用降维的思想，把多指标转化为少数几个综合指标。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from scipy.io import loadmat
```

```
data = loadmat('data/ex7data1.mat')
# data
```

```
X = data['X']

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(X[:, 0], X[:, 1])
plt.show()
```



PCA的算法相当简单。 在确保数据被归一化之后，输出仅仅是原始数据的协方差矩阵的奇异值分解。

```
def pca(X):
    # normalize the features
    X = (X - X.mean()) / X.std()

    # compute the covariance matrix
    X = np.matrix(X)
    cov = (X.T * X) / X.shape[0]

    # perform SVD
    U, S, V = np.linalg.svd(cov)

    return U, S, V
```

```
U, S, V = pca(X)
U, S, V
```

```
(matrix([[-0.79241747, -0.60997914],
         [-0.60997914,  0.79241747]]),
 array([1.43584536, 0.56415464]),
 matrix([[-0.79241747, -0.60997914],
         [-0.60997914,  0.79241747]]))
```

现在我们有主成分（矩阵U），我们可以用这些来将原始数据投影到一个较低维的空间中。 对于这个任务，我们将实现一个计算投影并且仅选择顶部K个分量的函数，有效地减少了维数。

```
def project_data(X, U, k):
    U_reduced = U[:,:k]
    return np.dot(X, U_reduced)
```

```
Z = project_data(X, U, 1)
Z
```

```
matrix([[-4.74689738],
        [-7.15889408],
        [-4.79563345],
        [-4.45754509],
        [-4.80263579],
        [-7.04081342],
        [-4.97025076],
        [-8.75934561],
        [-6.2232703 ],
        [-7.04497331],
        [-6.91702866],
        [-6.79543508],
        [-6.3438312 ],
        [-6.99891495],
        [-4.54558119],
        [-8.31574426],
        [-7.16920841],
        [-5.08083842],
        [-8.54077427],
        [-6.94102769],
        [-8.5978815 ],
        [-5.76620067],
        [-8.2020797 ],
        [-6.23890078],
        [-4.37943868],
        [-5.56947441],
        [-7.53865023],
        [-7.70645413],
        [-5.17158343],
        [-6.19268884],
        [-6.24385246],
        [-8.02715303],
        [-4.81235176],
        [-7.07993347],
```

```
        [-5.45953289],
        [-7.60014707],
        [-4.39612191],
        [-7.82288033],
        [-3.40498213],
        [-6.54290343],
        [-7.17879573],
        [-5.22572421],
        [-4.83081168],
        [-7.23907851],
        [-4.36164051],
        [-6.44590096],
        [-2.69118076],
        [-4.61386195],
        [-5.88236227],
        [-7.76732508]])
```

我们也可以通过反向转换步骤来恢复原始数据。

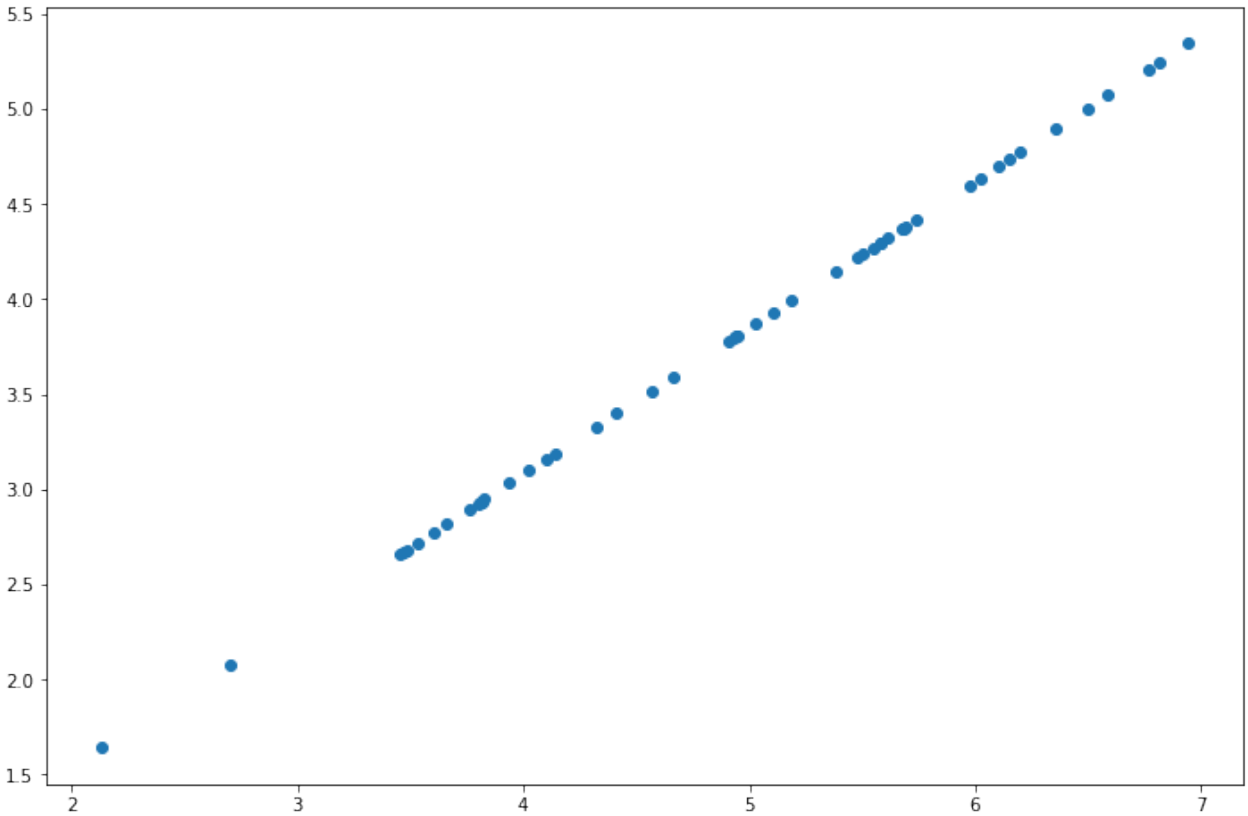
```
def recover_data(Z, U, k):
    U_reduced = U[:,:k]
    return np.dot(Z, U_reduced.T)
```

```
X_recovered = recover_data(Z, U, 1)
X_recovered
```

```
matrix([[3.76152442, 2.89550838],
        [5.67283275, 4.36677606],
        [3.80014373, 2.92523637],
        [3.53223661, 2.71900952],
        [3.80569251, 2.92950765],
        [5.57926356, 4.29474931],
        [3.93851354, 3.03174929],
        [6.94105849, 5.3430181 ],
        [4.93142811, 3.79606507],
        [5.58255993, 4.29728676],
        [5.48117436, 4.21924319],
        [5.38482148, 4.14507365],
        [5.02696267, 3.8696047 ],
        [5.54606249, 4.26919213],
        [3.60199795, 2.77270971],
        [6.58954104, 5.07243054],
        [5.681006  , 4.37306758],
        [4.02614513, 3.09920545],
```

```
        [6.76785875, 5.20969415],
        [5.50019161, 4.2338821 ],
        [6.81311151, 5.24452836],
        [4.56923815, 3.51726213],
        [6.49947125, 5.00309752],
        [4.94381398, 3.80559934],
        [3.47034372, 2.67136624],
        [4.41334883, 3.39726321],
        [5.97375815, 4.59841938],
        [6.10672889, 4.70077626],
        [4.09805306, 3.15455801],
        [4.90719483, 3.77741101],
        [4.94773778, 3.80861976],
        [6.36085631, 4.8963959 ],
        [3.81339161, 2.93543419],
        [5.61026298, 4.31861173],
        [4.32622924, 3.33020118],
        [6.02248932, 4.63593118],
        [3.48356381, 2.68154267],
        [6.19898705, 4.77179382],
        [2.69816733, 2.07696807],
        [5.18471099, 3.99103461],
        [5.68860316, 4.37891565],
        [4.14095516, 3.18758276],
        [3.82801958, 2.94669436],
        [5.73637229, 4.41568689],
        [3.45624014, 2.66050973],
        [5.10784454, 3.93186513],
        [2.13253865, 1.64156413],
        [3.65610482, 2.81435955],
        [4.66128664, 3.58811828],
        [6.1549641 , 4.73790627]])
```

```
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(list(X_recovered[:, 0]), list(X_recovered[:, 1]))
plt.show()
```



请注意，第一主成分的投影轴基本上是数据集中的对角线。 当我们将数据减少到一个维度时，我们失去了该对角线周围的变化，所以在我们的再现中，一切都沿着该对角线。

本文代码更新地址：https://github.com/fengdu78/lihang-code

中文注释制作：机器学习初学者公众号：ID:ai-start-com

配置环境：python 3.5+

代码全部测试通过。