

线性代数

朱明超

Email: deityrayleigh@gmail.com

Github: github.com/MingchaoZhu/DeepLearning

1 标量, 向量, 矩阵, 张量

1. **标量 (Scalar)**: 表示一个单独的数, 通常用斜体小写字母表示, 如 $s \in \mathbb{R}, n \in \mathbb{N}$ 。

2. **向量 (Vector)**: 表示一列数, 这些数有序排列的, 可以通过下标获取对应值, 通常用粗体小写字母表示: $\mathbf{x} \in \mathbb{R}^n$, 它表示元素取实数, 且有 n 个元素, 第一个元素表示为: x_1 。将向量写成列向量的形式:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad (1)$$

有时需要向量的子集, 例如第 1, 3, 6 个元素, 那么我们可以令集合 $S = \{1, 3, 6\}$, 然后用 \mathbf{x}_S 来表示这个子集。另外, 我们用符号 $-$ 表示集合的补集: \mathbf{x}_{-1} 表示除 x_1 外 \mathbf{x} 中的所有元素, \mathbf{x}_{-S} 表示除 x_1, x_3, x_6 外 \mathbf{x} 中的所有元素。

3. **矩阵 (Matrix)**: 表示一个二维数组, 每个元素的下标由两个数字确定, 通常用大写粗体字母表示: $\mathbf{A} \in \mathbb{R}^{m \times n}$, 它表示元素取实数的 m 行 n 列矩阵, 其元素可以表示为: $A_{1,1}, A_{m,n}$ 。我们用 $:$ 表示矩阵的一行或者一列: $\mathbf{A}_{i,:}$ 为第 i 行, $\mathbf{A}_{:,j}$ 为第 j 列。

矩阵可以写成这样的形式:

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \quad (2)$$

有时我们需要对矩阵进行逐元素操作, 如将函数 f 应用到 \mathbf{A} 的所有元素上, 此时我们用 $f(\mathbf{A})_{i,j}$ 表示。

4. **张量 (Tensor)**: 超过二维的数组, 我们用 \mathbf{A} 表示张量, $A_{i,j,k}$ 表示其元素 (三维张量情况下)。

```
[1]: import numpy as np
```

```
[2]: # 标量
s = 5
# 向量
v = np.array([1,2])
# 矩阵
m = np.array([[1,2], [3,4]])
# 张量
t = np.array([
    [[1,2,3], [4,5,6], [7,8,9]],
    [[11,12,13], [14,15,16], [17,18,19]],
    [[21,22,23], [24,25,26], [27,28,29]],
])
print("标量: " + str(s))
print("向量: " + str(v))
print("矩阵: " + str(m))
print("张量: " + str(t))
```

标量: 5

向量: [1 2]

矩阵: [[1 2]

[3 4]]

张量: [[[1 2 3]

[4 5 6]

[7 8 9]]

[[11 12 13]

[14 15 16]

[17 18 19]]

```
[[21 22 23]
 [24 25 26]
 [27 28 29]]]
```

2 矩阵转置

矩阵转置 (Transpose) 相当于沿着对角线翻转，定义如下：

$$A_{i,j}^T = A_{i,j} \quad (3)$$

矩阵转置的转置等于矩阵本身：

$$\left(A^T\right)^T = A \quad (4)$$

转置将矩阵的形状从 $m \times n$ 变成了 $n \times m$ 。

向量可以看成是只有一列的矩阵，为了方便，我们可以使用行向量加转置的操作，如： $\mathbf{x} = [x_1, x_2, x_3]^T$ 。

标量也可以看成是一行一列的矩阵，其转置等于它自身： $a^T = a$ 。

```
[3]: A = np.array([[1.0,2.0],[1.0,0.0],[2.0,3.0]])
A_t = A.transpose()
print("A:", A)
print("A 的转置:", A_t)
```

```
A: [[1. 2.]
 [1. 0.]
 [2. 3.]]
A 的转置: [[1. 1. 2.]
 [2. 0. 3.]]
```

3 矩阵加法

加法即对应元素相加，要求两个矩阵的形状一样：

$$C = A + B, C_{i,j} = A_{i,j} + B_{i,j} \quad (5)$$

数乘即一个标量与矩阵每个元素相乘：

$$D = a \cdot B + c, D_{i,j} = a \cdot B_{i,j} + c \quad (6)$$

有时我们允许矩阵和向量相加的，得到一个矩阵，把 \mathbf{b} 加到了 A 的每一行上，本质上是构造了一个将 \mathbf{b} 按行复制的一个新矩阵，这种机制叫做广播 (Broadcasting)：

$$C = A + \mathbf{b}, C_{i,j} = A_{i,j} + b_j \quad (7)$$

```
[4]: a = np.array([[1.0,2.0],[3.0,4.0]])
b = np.array([[6.0,7.0],[8.0,9.0]])
print("矩阵相加: ", a + b)
```

```
矩阵相加: [[ 7.  9.]
 [11. 13.]]
```

4 矩阵乘法

两个矩阵相乘得到第三个矩阵，我们需要 A 的形状为 $m \times n$ ， B 的形状为 $n \times p$ ，得到的矩阵为 C 的形状为 $m \times p$ ：

$$C = AB \quad (8)$$

具体定义为

$$C_{i,j} = \sum_k A_{i,k} B_{k,j} \quad (9)$$

注意矩阵乘法不是元素对应相乘，元素对应相乘又叫 Hadamard 乘积，记作 $A \odot B$ 。

向量可以看作是列为 1 的矩阵，两个相同维数的向量 \mathbf{x} 和 \mathbf{y} 的点乘 (Dot Product) 或者内积，可以表示为 $\mathbf{x}^T \mathbf{y}$ 。

我们也可以把矩阵乘法理解为： $C_{i,j}$ 表示 A 的第 i 行与 B 的第 j 列的点积。

```
[5]: m1 = np.array([[1.0,3.0],[1.0,0.0]])
      m2 = np.array([[1.0,2.0],[5.0,0.0]])
      print("按矩阵乘法规则: ", np.dot(m1, m2))
      print("按逐元素相乘: ", np.multiply(m1, m2))
      print("按逐元素相乘: ", m1*m2)
      v1 = np.array([1.0,2.0])
      v2 = np.array([4.0,5.0])
      print("向量内积: ", np.dot(v1, v2))
```

```
按矩阵乘法规则: [[16.  2.]
 [ 1.  2.]]
按逐元素相乘: [[1.  6.]
 [5.  0.]]
按逐元素相乘: [[1.  6.]
 [5.  0.]]
向量内积: 14.0
```

5 单位矩阵

为了引入矩阵的逆，我们需要先定义单位矩阵 (Identity Matrix): 单位矩阵乘以任意一个向量等于这个向量本身。记 I_n 为保持 n 维向量不变的单位矩阵，即：

$$I_n \in \mathbb{R}^{n \times n}, \forall \mathbf{x} \in \mathbb{R}^n, I_n \mathbf{x} = \mathbf{x} \quad (10)$$

单位矩阵的结构十分简单，所有的对角元素都为 1，其他元素都为 0，如：

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (11)$$

```
[6]: np.identity(3)
```

```
[6]: array([[1., 0., 0.],
           [0., 1., 0.],
           [0., 0., 1.]])
```

6 矩阵的逆

矩阵 A 的逆 (Inversion) 记作 A^{-1} ，定义为一个矩阵使得

$$A^{-1}A = I_n \quad (12)$$

如果 A^{-1} 存在，那么线性方程组 $A\mathbf{x} = \mathbf{b}$ 的解为：

$$A^{-1}A\mathbf{x} = I_n\mathbf{x} = \mathbf{x} = A^{-1}\mathbf{b} \quad (13)$$

```
[7]: A = [[1.0,2.0],[3.0,4.0]]
      A_inv = np.linalg.inv(A)
      print("A 的逆矩阵", A_inv)
```

```
A 的逆矩阵 [[-2.  1.]
 [ 1.5 -0.5]]
```

7 范数

通常我们用范数 (norm) 来衡量向量，向量的 L^p 范数定义为：

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}, p \in \mathbb{R}, p \geq 1 \quad (14)$$

L^2 范数，也称欧几里得范数 (Euclidean norm)，是向量 \mathbf{x} 到原点的欧几里得距离。有时也用 L^2 范数的平方来衡量向量： $\mathbf{x}^T \mathbf{x}$ 。事实上，平方 L^2 范数在计算上更为便利，例如它的对 \mathbf{x} 梯度的各个分量只依赖于 \mathbf{x} 的对应的各个分量，而 L^2 范数对 \mathbf{x} 梯度的各个分量要依赖于整个 \mathbf{x} 向量。

L^1 范数: L^2 范数并不一定适用于所有的情况, 它在原点附近的生长就十分缓慢, 因此不适用于需要区别 0 和非常小但是非 0 值的情况。 L^1 范数就是一个比较好的选择, 它在所有方向上的增长速率都是一样的, 定义为:

$$\|\mathbf{x}\|_1 = \sum_i |x_i| \quad (15)$$

它经常使用在需要区分 0 和非 0 元素的情形中。

L^0 范数: 如果需要衡量向量中非 0 元素的个数, 但它并不是一个范数 (不满足三角不等式和数乘), 此时 L^1 范数可以作为它的一个替代。

L^∞ 范数: 它在数学上是向量元素绝对值的最大值, 因此也被叫做 (Max norm):

$$\|\mathbf{x}\|_\infty = \max_i |x_i| \quad (16)$$

有时我们想衡量一个矩阵, 机器学习中通常使用的是 F 范数 (Frobenius norm), 其定义为:

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} A_{i,j}^2} \quad (17)$$

```
[8]: a = np.array([1.0,3.0])
      print("向量 2 范数", np.linalg.norm(a,ord=2))
      print("向量 1 范数", np.linalg.norm(a,ord=1))
      print("向量无穷范数", np.linalg.norm(a,ord=np.inf))
```

向量 2 范数 3.1622776601683795

向量 1 范数 4.0

向量无穷范数 3.0

```
[9]: a = np.array([[1.0,3.0],[2.0,1.0]])
      print("矩阵 F 范数", np.linalg.norm(a,ord="fro"))
```

矩阵 F 范数 3.872983346207417

8 特征值分解

如果一个 $n \times n$ 矩阵 \mathbf{A} 有 n 组线性无关的单位特征向量 $\{\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(n)}\}$, 以及对应的特征值 $\lambda_1, \dots, \lambda_n$ 。将这些特征向量按列拼接成一个矩阵: $\mathbf{V} = [\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(n)}]$, 并将对应的特征值拼接成一个向量: $\boldsymbol{\lambda} = [\lambda_1, \dots, \lambda_n]$ 。

\mathbf{A} 的特征值分解 (Eigendecomposition) 为:

$$\mathbf{A} = \mathbf{V} \text{diag}(\boldsymbol{\lambda}) \mathbf{V}^{-1} \quad (18)$$

注意:

- 不是所有的矩阵都有特征值分解
- 在某些情况下, 实矩阵的特征值分解可能会得到复矩阵

```
[10]: A = np.array([[1.0,2.0,3.0],
                   [4.0,5.0,6.0],
                   [7.0,8.0,9.0]])
      # 计算特征值
      print("特征值:", np.linalg.eigvals(A))
      # 计算特征值和特征向量
      eigvals,eigvectors = np.linalg.eig(A)
      print("特征值:", eigvals)
      print("特征向量:", eigvectors)
```

特征值: [1.61168440e+01 -1.11684397e+00 -3.73313677e-16]

特征值: [1.61168440e+01 -1.11684397e+00 -3.73313677e-16]

特征向量: [[-0.23197069 -0.78583024 0.40824829]

[-0.52532209 -0.08675134 -0.81649658]

[-0.8186735 0.61232756 0.40824829]]

9 奇异值分解

奇异值分解 (Singular Value Decomposition, SVD) 提供了另一种分解矩阵的方式, 将其分解为奇异向量和奇异值。

与特征值分解相比, 奇异值分解更加通用, 所有的实矩阵都可以进行奇异值分解, 而特征值分解只对某些方阵可以。

奇异值分解的形式为:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (19)$$

若 \mathbf{A} 是 $m \times n$ 的, 那么 \mathbf{U} 是 $m \times m$ 的, 其列向量称为左奇异向量, 而 \mathbf{V} 是 $n \times n$ 的, 其列向量称为右奇异向量, 而 $\mathbf{\Sigma}$ 是 $m \times n$ 的一个对角矩阵, 其对角元素称为矩阵 \mathbf{A} 的奇异值。

事实上, 左奇异向量是 $\mathbf{A}\mathbf{A}^T$ 的特征向量, 而右奇异向量是 $\mathbf{A}^T\mathbf{A}$ 的特征向量, 非 0 奇异值的平方是 $\mathbf{A}^T\mathbf{A}$ 的非 0 特征值。

```
[11]: A = np.array([[1.0,2.0,3.0],
                  [4.0,5.0,6.0]])
U,D,V = np.linalg.svd(A)
print("U:", U)
print("D:", D)
print("V:", V)
```

```
U: [[-0.3863177 -0.92236578]
     [-0.92236578  0.3863177 ]]
D: [9.508032  0.77286964]
V: [[-0.42866713 -0.56630692 -0.7039467 ]
     [ 0.80596391  0.11238241 -0.58119908]
     [ 0.40824829 -0.81649658  0.40824829]]
```

10 PCA (主成分分析)

假设我们有 m 个数据点 $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)} \in \mathbb{R}^n$, 对于每个数据点 $\mathbf{x}^{(i)}$, 我们希望找到一个对应的点 $\mathbf{c}^{(i)} \in \mathbb{R}^l, l < n$ 去表示它 (相当于对它进行降维), 并且让损失的信息量尽可能少。

我们可以将这个过程看作是一个编码解码的过程, 设编码和解码函数分别为 f, g , 则有 $f(\mathbf{x}) = \mathbf{c}, \mathbf{x} \approx g(f(\mathbf{x}))$ 。考虑一个线性解码函数 $g(\mathbf{c}) = \mathbf{D}\mathbf{c}, \mathbf{D} \in \mathbb{R}^{n \times l}$, 为了计算方便, 我们将这个矩阵的列向量约束为相互正交的。另一方面, 考虑到存在尺度放缩的问题, 我们将这个矩阵的列向量约束为具有单位范数来获得唯一解。

对于给定的 \mathbf{x} , 我们需要找到信息损失最小的 \mathbf{c}^* , 即求解:

$$\mathbf{c}^* = \arg \min_{\mathbf{c}} \|\mathbf{x} - g(\mathbf{c})\|_2 = \arg \min_{\mathbf{c}} \|\mathbf{x} - g(\mathbf{c})\|_2^2 \quad (20)$$

这里我们用二范数来衡量信息的损失。展开之后我们有:

$$\|\mathbf{x} - g(\mathbf{c})\|_2^2 = (\mathbf{x} - g(\mathbf{c}))^T (\mathbf{x} - g(\mathbf{c})) = \mathbf{x}^T \mathbf{x} - 2\mathbf{x}^T g(\mathbf{c}) + g(\mathbf{c})^T g(\mathbf{c}) \quad (21)$$

结合 $g(\mathbf{c})$ 的表达式, 忽略不依赖 \mathbf{c} 的 $\mathbf{x}^T \mathbf{x}$ 项, 我们有:

$$\begin{aligned} \mathbf{c}^* &= \arg \min_{\mathbf{c}} -2\mathbf{x}^T \mathbf{D}\mathbf{c} + \mathbf{c}^T \mathbf{D}^T \mathbf{D}\mathbf{c} \\ &= \arg \min_{\mathbf{c}} -2\mathbf{x}^T \mathbf{D}\mathbf{c} + \mathbf{c}^T \mathbf{I}_l \mathbf{c} \\ &= \arg \min_{\mathbf{c}} -2\mathbf{x}^T \mathbf{D}\mathbf{c} + \mathbf{c}^T \mathbf{c} \end{aligned} \quad (22)$$

这里 \mathbf{D} 具有单位正交性。

对 \mathbf{c} 求梯度, 并令其为零, 我们有:

$$\begin{aligned} \nabla_{\mathbf{c}}(-2\mathbf{x}^T \mathbf{D}\mathbf{c} + \mathbf{c}^T \mathbf{c}) &= \mathbf{0} \\ -2\mathbf{D}^T \mathbf{x} + 2\mathbf{c} &= \mathbf{0} \\ \mathbf{c} &= \mathbf{D}^T \mathbf{x} \end{aligned} \quad (23)$$

因此, 我们的编码函数为:

$$f(\mathbf{x}) = \mathbf{D}^T \mathbf{x} \quad (24)$$

此时通过编码解码得到的重构为:

$$r(\mathbf{x}) = g(f(\mathbf{x})) = \mathbf{D}\mathbf{D}^T \mathbf{x} \quad (25)$$

接下来求解最优的变换 D 。由于我们需要将 D 应用到所有的 \mathbf{x}_i 上，所以我们需要最优化：

$$\begin{aligned} D^* &= \arg \min_D \sqrt{\sum_{i,j} (\mathbf{x}_j^{(i)} - r(\mathbf{x}^{(i)})_j)^2} \\ \text{s.t. } D^\top D &= I_l \end{aligned} \quad (26)$$

为了方便，我们考虑 $l = 1$ 的情况，此时问题简化为：

$$\begin{aligned} \mathbf{d}^* &= \arg \min_{\mathbf{d}} \sum_i (\mathbf{x}_j^{(i)} - \mathbf{d}\mathbf{d}^\top \mathbf{x}^{(i)})^2 \\ \text{s.t. } \mathbf{d}^\top \mathbf{d} &= 1 \end{aligned} \quad (27)$$

考虑 F 范数，并进一步的推导：

$$\begin{aligned} \mathbf{d}^* &= \arg \max_{\mathbf{d}} \text{Tr}(\mathbf{d}^\top \mathbf{X}^\top \mathbf{X} \mathbf{d}) \\ \text{s.t. } \mathbf{d}^\top \mathbf{d} &= 1 \end{aligned} \quad (28)$$

优化问题可以用特征分解来求解。但实际计算时，我们会采用如下方式计算：

PCA 将输入 \mathbf{x} 投影表示成 \mathbf{c} 。 \mathbf{c} 是比原始输入维数更低的表示，同时使得元素之间线性无关。假设有一个 $m \times n$ 的矩阵 \mathbf{X} ，数据的均值为零，即 $\mathbb{E}[\mathbf{x}] = 0$ ， \mathbf{X} 对应的无偏样本协方差矩阵： $\text{Var}[\mathbf{x}] = \frac{1}{m-1} \mathbf{X}^\top \mathbf{X}$ 。

PCA 是通过线性变换找到一个 $\text{Var}[\mathbf{c}]$ 是对角矩阵的表示 $\mathbf{c} = \mathbf{V}^\top \mathbf{x}$ ，矩阵 \mathbf{X} 的主成分可以通过奇异值分解 (SVD) 得到，也就是说主成分是 \mathbf{X} 的右奇异向量。假设 \mathbf{V} 是 $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ 奇异值分解的右奇异向量，我们得到原来的特征向量方程：

$$\mathbf{X}^\top \mathbf{X} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top)^\top \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{V}\mathbf{\Sigma}^\top \mathbf{U}^\top \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{V}\mathbf{\Sigma}^2 \mathbf{V}^\top \quad (29)$$

因为根据奇异值的定义 $\mathbf{U}^\top \mathbf{U} = \mathbf{I}$ 。因此 \mathbf{X} 的方差可以表示为： $\text{Var}[\mathbf{x}] = \frac{1}{m-1} \mathbf{X}^\top \mathbf{X} = \frac{1}{m-1} \mathbf{V}\mathbf{\Sigma}^2 \mathbf{V}^\top$ 。

所以 \mathbf{c} 的协方差满足： $\text{Var}[\mathbf{c}] = \frac{1}{m-1} \mathbf{C}^\top \mathbf{C} = \frac{1}{m-1} \mathbf{V}^\top \mathbf{X}^\top \mathbf{X} \mathbf{V} = \frac{1}{m-1} \mathbf{V}^\top \mathbf{V}\mathbf{\Sigma}^2 \mathbf{V}^\top \mathbf{V} = \frac{1}{m-1} \mathbf{\Sigma}^2$ ，因为根据奇异值定义 $\mathbf{V}^\top \mathbf{V} = \mathbf{I}$ 。 \mathbf{c} 的协方差是对角的， \mathbf{c} 中的元素是彼此无关的。

以 *iris* 数据为例，展示 PCA 的使用。

```
[12]: import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
%matplotlib inline
```

```
[13]: # 载入数据
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['label'] = iris.target
df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
df.label.value_counts()
```

```
[13]: 2    50
      1    50
      0    50
      Name: label, dtype: int64
```

```
[14]: # 查看数据
df.tail()
```

```
[14]:   sepal length  sepal width  petal length  petal width  label
145         6.7         3.0         5.2         2.3         2
146         6.3         2.5         5.0         1.9         2
147         6.5         3.0         5.2         2.0         2
148         6.2         3.4         5.4         2.3         2
149         5.9         3.0         5.1         1.8         2
```

```
[15]: # 查看数据
X = df.iloc[:, 0:4]
y = df.iloc[:, 4]
print("查看第一个数据: \n", X.iloc[0, 0:4])
print("查看第一个标签: \n", y.iloc[0])
```

查看第一个数据：

```
sepal length    5.1
sepal width     3.5
petal length    1.4
petal width     0.2
Name: 0, dtype: float64
```

查看第一个标签：

0

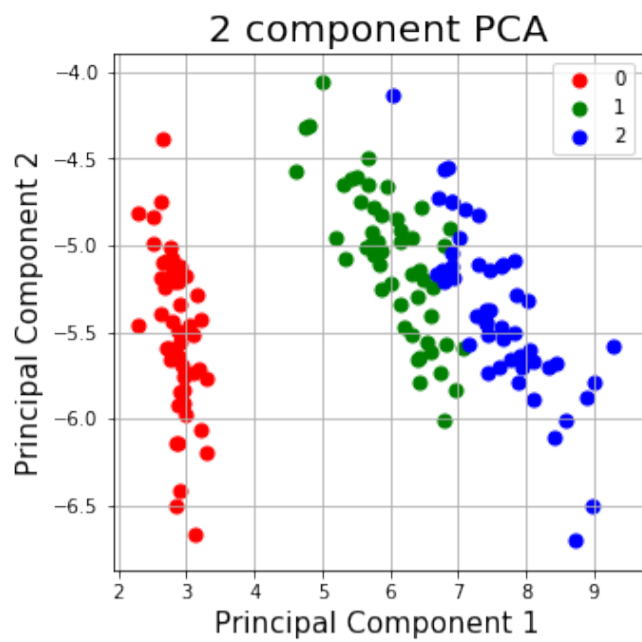
```
[16]: class PCA():
        def __init__(self):
            pass

        def fit(self, X, n_components):
            n_samples = np.shape(X)[0]
            covariance_matrix = (1 / (n_samples-1)) * (X - X.mean(axis=0)).T.dot(X - X.mean(axis=0))
            # 对协方差矩阵进行特征值分解
            eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
            # 对特征值（特征向量）从大到小排序
            idx = eigenvalues.argsort()[::-1]
            eigenvalues = eigenvalues[idx][:n_components]
            eigenvectors = np.atleast_1d(eigenvectors[:, idx])[:, :n_components]
            # 得到低维表示
            X_transformed = X.dot(eigenvectors)
            return X_transformed
```

```
[17]: model = PCA()
        Y = model.fit(X, 2)
```

```
[18]: principalDf = pd.DataFrame(np.array(Y),
                                columns=['principal component 1', 'principal component 2'])
        Df = pd.concat([principalDf, y], axis = 1)
        fig = plt.figure(figsize = (5,5))
        ax = fig.add_subplot(1,1,1)
        ax.set_xlabel('Principal Component 1', fontsize = 15)
        ax.set_ylabel('Principal Component 2', fontsize = 15)
        ax.set_title('2 component PCA', fontsize = 20)

        targets = [0, 1, 2]
        # ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
        colors = ['r', 'g', 'b']
        for target, color in zip(targets, colors):
            indicesToKeep = Df['label'] == target
            ax.scatter(Df.loc[indicesToKeep, 'principal component 1']
                      , Df.loc[indicesToKeep, 'principal component 2']
                      , c = color
                      , s = 50)
        ax.legend(targets)
        ax.grid()
```

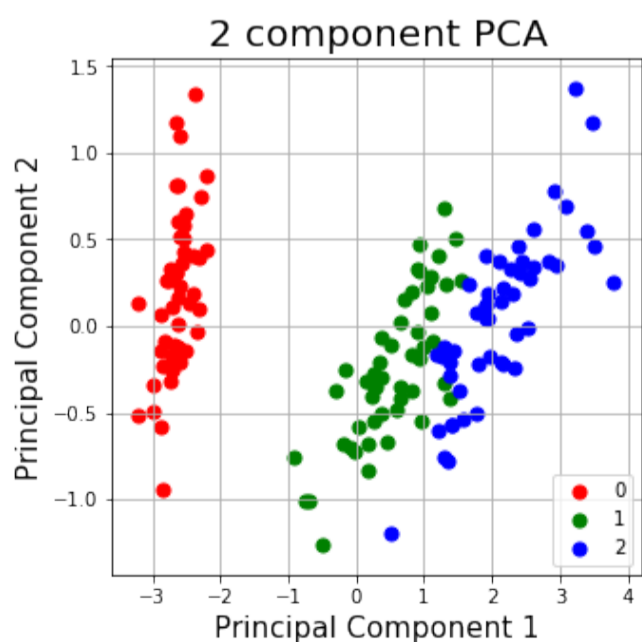


使用 *sklearn* 包实现 PCA

```
[19]: from sklearn.decomposition import PCA as sklearnPCA
sklearn_pca = sklearnPCA(n_components=2)
Y = sklearn_pca.fit_transform(X)
```

```
[20]: principalDf = pd.DataFrame(data = np.array(Y), columns = ['principal component 1', 'principal component 2'])
Df = pd.concat([principalDf, y], axis = 1)
fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

targets = [0, 1, 2]
# ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets, colors):
    indicesToKeep = Df['label'] == target
    ax.scatter(Df.loc[indicesToKeep, 'principal component 1']
              , Df.loc[indicesToKeep, 'principal component 2']
              , c = color
              , s = 50)
ax.legend(targets)
ax.grid()
```



```
[21]: import numpy, pandas, matplotlib, sklearn
print("numpy:", numpy.__version__)
print("pandas:", pandas.__version__)
```



```
print("matplotlib:", matplotlib.__version__)  
print("sklearn:", sklearn.__version__)
```

numpy: 1.14.5

pandas: 0.25.1

matplotlib: 3.1.1

sklearn: 0.21.3