Numpy进阶-HITS算法 C07

信息科学技术学院

胡俊峰

主要内容

- Numpy的条件筛选
- 特征值与特征向量
- 信息熵、主成分分解、特征空间变换与数据降维
- SVD-HITS与推荐算法

numpy.where

numpy.where(*condition*, [*x*, *y*,]/)

Return elements chosen from *x* or *y* depending on *condition*.

> **Note**
>
> When only *condition* is provided, this function is a shorthand for `np.asarray(condition).nonzero()`. Using `nonzero` directly should be preferred, as it behaves correctly for subclasses. The rest of this documentation covers only the case where all three arguments are provided.

Parameters: **condition** : *array_like, bool*

Where True, yield *x*, otherwise yield *y*.

x, y : *array_like*

Values from which to choose. *x*, *y* and *condition* need to be broadcastable to shape.

Returns: **out** : *ndarray*

An array with elements from *x* where *condition* is True, and elements from *y* elsewhere.

Notes

If all the arrays are 1-D, `where` is equivalent to:

```
[xv if c else yv
 for c, xv, yv in zip(condition, x, y)]
```

Examples

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.where(a < 5, a, 10*a)
array([ 0,  1,  2,  3,  4, 50, 60, 70, 80, 90])
```

用where()函数返回（布尔）下标

```
y = np.array([1,5,6,8,1,7,3,6,9])# Where y is greater than 5, ret
print(np.where(y>5))
print(np.where(y>5 ,0,1))  # 返回布尔下标
z = np.array(y[np.where(y>5)])
z
```

```
(array([2, 3, 5, 7, 8], dtype=int64),)
[1 1 0 0 1 0 1 0 0]

array([6, 8, 7, 6, 9])
```

numpy.extract

`numpy.extract(condition, arr)` [source]

Return the elements of an array that satisfy some condition.

This is equivalent to `np.compress(ravel(condition), ravel(arr))`. If *condition* is boolean `np.extract` is equivalent to `arr[condition]`.

Note that `place` does the exact opposite of `extract`.

Parameters: **condition** : *array_like*

An array whose nonzero or True entries indicate the elements of *arr* to extract.

arr : *array_like*

Input array of the same size as *condition*.

Returns: **extract** : *ndarray*

Rank 1 array of values from *arr* where *condition* is True.

```python
arr = np.arange(12).reshape((3, 4))
arr
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```python
np.extract(arr>4, arr)
```

```
array([ 5,  6,  7,  8,  9, 10, 11])
```

```python
np.where(arr>4)
```

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

clip(): 裁剪

```
# np.clip(a, a_min, a_max, out=None)
a = np.random.randint(13, size =8)
a
```

```
array([ 0,  1, 11,  6,  5,  6,  7,  2])
```

```
np.clip(a, 1, 9)
```

```
array([1, 1, 9, 6, 5, 6, 7, 2])
```

把数组保存到文件与读取数组文件:

```
# 文件保存方式:  save ,savetext,savez_compressed
ar1 = np.arange(10)
ar2 = np.arange(-5,6,2).reshape(2,3)
np.save('some_array',ar1)          # 保存数组
ar3 = np.load('some_array.npy')   # 读取数组
print(ar3)
np.savez('array_archive.npz',a = ar1,b =ar2) # 可将多个数组保存到一个未压缩文件中
zipfiles = np.load('array_archive.npz')
print(zipfiles.files)
ar6 = zipfiles['b']
ar6
```

```
[0 1 2 3 4 5 6 7 8 9]
['a', 'b']

array([[-5, -3, -1],
       [ 1,  3,  5]])
```

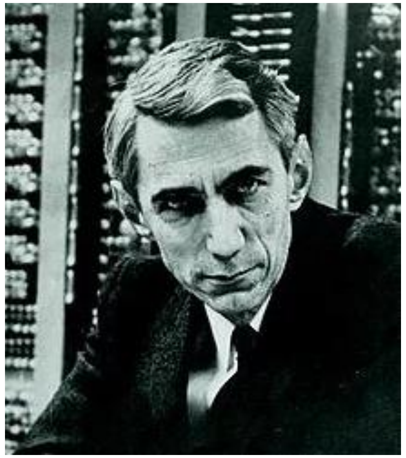
基于矩阵分解的数据特征分析

- 编码系统的信息熵
- 数据样本集的特征分解
- 特征空间聚类算法

A Mathematical Theory of Communication

By C. E. SHANNON

1948 香农



- 定义了'信息量'的数学定义
- 同时也界定了'学习过程'的计算表达

$$H = -(p \log p + q \log q)$$

in the case of two possibilities with probabilities p and $q = 1 - p$,

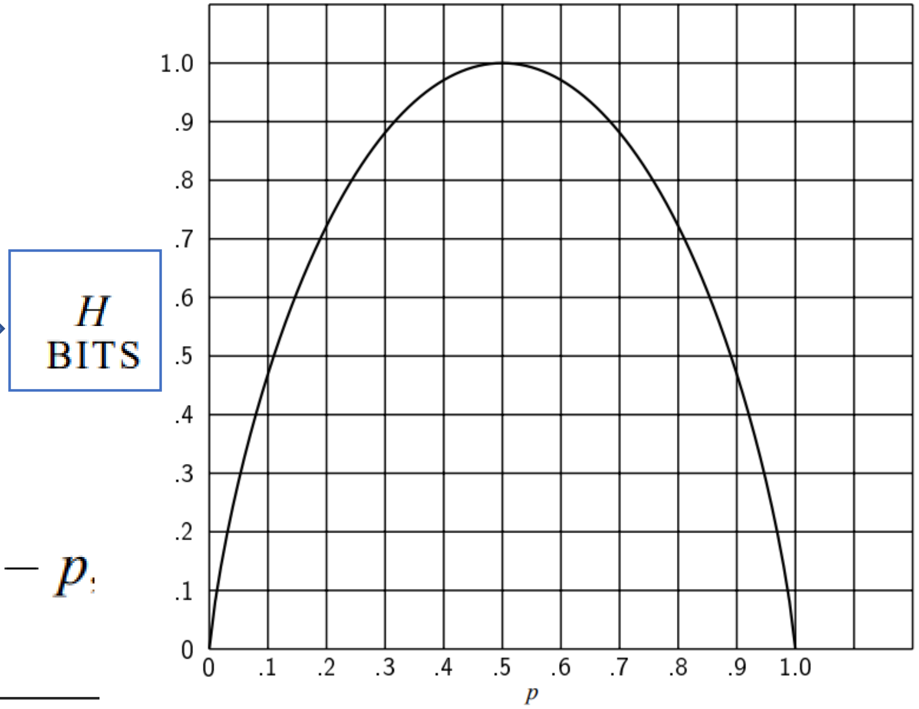


Fig. 7—Entropy in the case of two possibilities with probabilities p and $(1-p)$.

编码系统的信息量（信息熵）

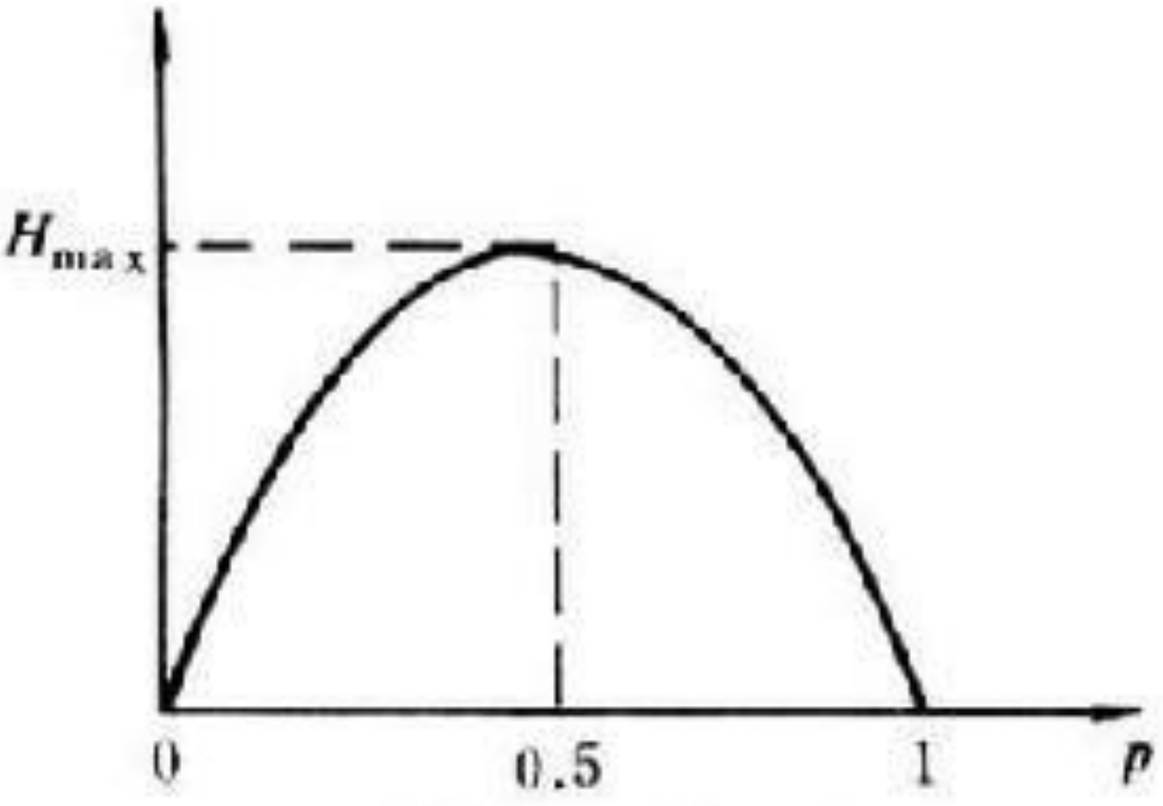
$$H(U) = E[-\log p_i] = -\sum_{i=1}^{n} p_i \log p_i$$

2元编码系统均匀分布的信息熵:

H2 = 2*（- 1/2 log（1/2））= 1 bit

4元编码系统均匀分布的信息熵

= 2bit

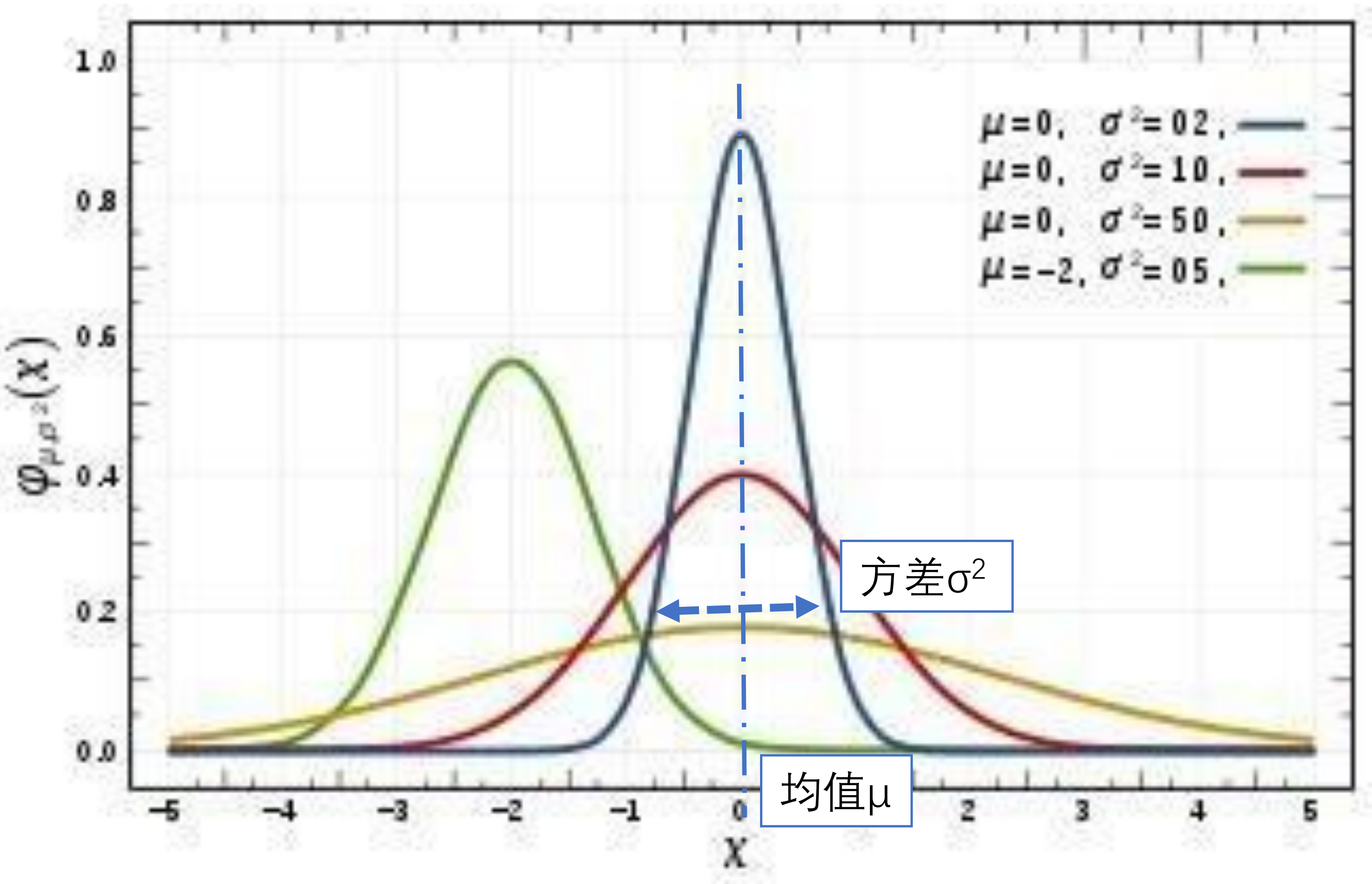


二元信源的熵函数

概率分布 - 方差 – 特征区分度

$$p(y)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(y-\mu)^2}{2\sigma^2}}$$

一个特征分布的方差越大，
其信息量即区分度也就越高



特征空间正交化、标准化

- 正交化：减少特征间信息冗余
- 标准化：统一特征的量纲（PCA后可以做）

- 中心化：获取特征的信息量

数据分布的中心化、标准化:

```python
# loc:均值 scale: 标准差;
sample = np.random.normal(loc=2., scale=1,size=(6, 2))#生成一个分布
sample
```

```
array([[2.00452325, 1.55588864],
       [0.93133888, 3.24697625],
       [1.11057341, 1.53323954],
       [1.89100717, 4.01663425],
       [3.10340241, 0.82610518],
       [2.48621024, 3.02368708]])
```

```python
# loc:均值 scale: 标准差;
#生成两个分布，对应两列数据
sample = np.random.normal(loc=[2., 20.], scale=[1., 4.5],size=(6, 2))
sample
```

```
array([[ 1.2234043 , 20.18968583],
       [ 1.41187264, 30.51115214],
       [ 2.445158  , 27.14954764],
       [ 2.3452225 , 21.94896694],
       [ 0.39418607, 21.4189724 ],
       [ 2.4348436 , 13.09173282]])
```

```python
mu = sample.mean(axis=0)   # 计算期望
mu
```

```
array([ 1.70911452, 22.38500963])
```

```python
# 中心化
print('sample:', sample.shape, '| means:', mu.shape) # 广播

sample - mu                # sample - sample.mean(axis=0)
```

```
sample: (6, 2) | means: (2,)
```

```
array([[-0.48571022, -2.1953238 ],
       [-0.29724188,  8.12614251],
       [ 0.73604348,  4.76453801],
       [ 0.63610798, -0.43604269],
       [-1.31492845, -0.96603723],
       [ 0.72572908, -9.2932768 ]])
```

```python
# z-score 的计算定义如下:
# z = (x-μ) /σ
std_sample = (sample - sample.mean(axis=0)) / sample.std(axis=0) # 标准化
std_sample
```

```
array([[-0.63356081, -0.39965345],
       [-0.38772255,  1.47934481],
       [ 0.96009573,  0.86737275],
       [ 0.82973978, -0.07938053],
       [-1.71519376, -0.17586477],
       [ 0.94664162, -1.69181881]])
```

```python
sample.min(axis=1)
```

```
array([1.2234043 , 1.41187264, 2.445158  , 2.3452225 , 0.39418607,
       2.4348436 ])
```

矩阵的特征值与特征向量

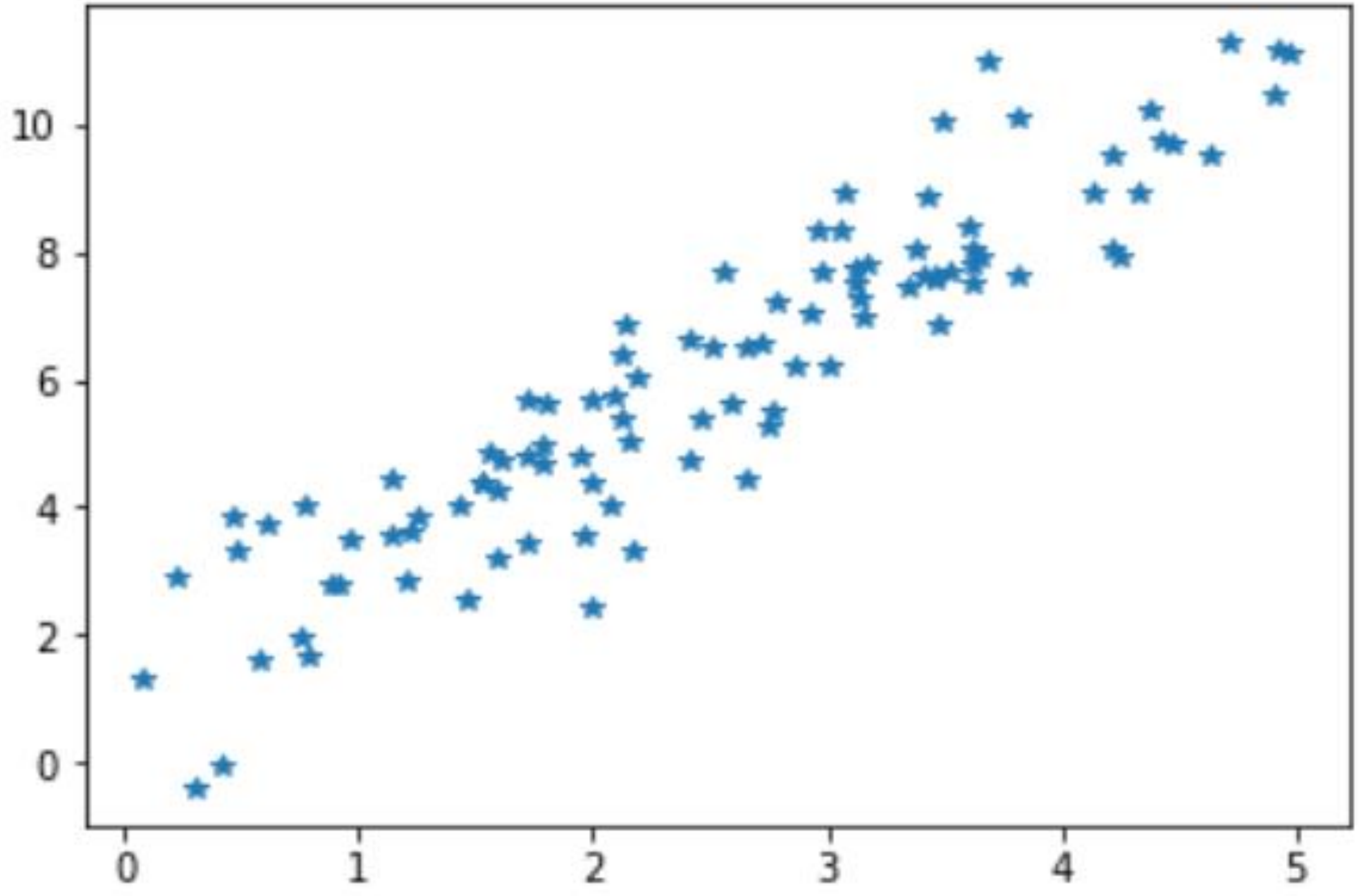
- 求特征值
- 特征向量的物理意义
- 特征空间变换与数据降维

生成二维数据:

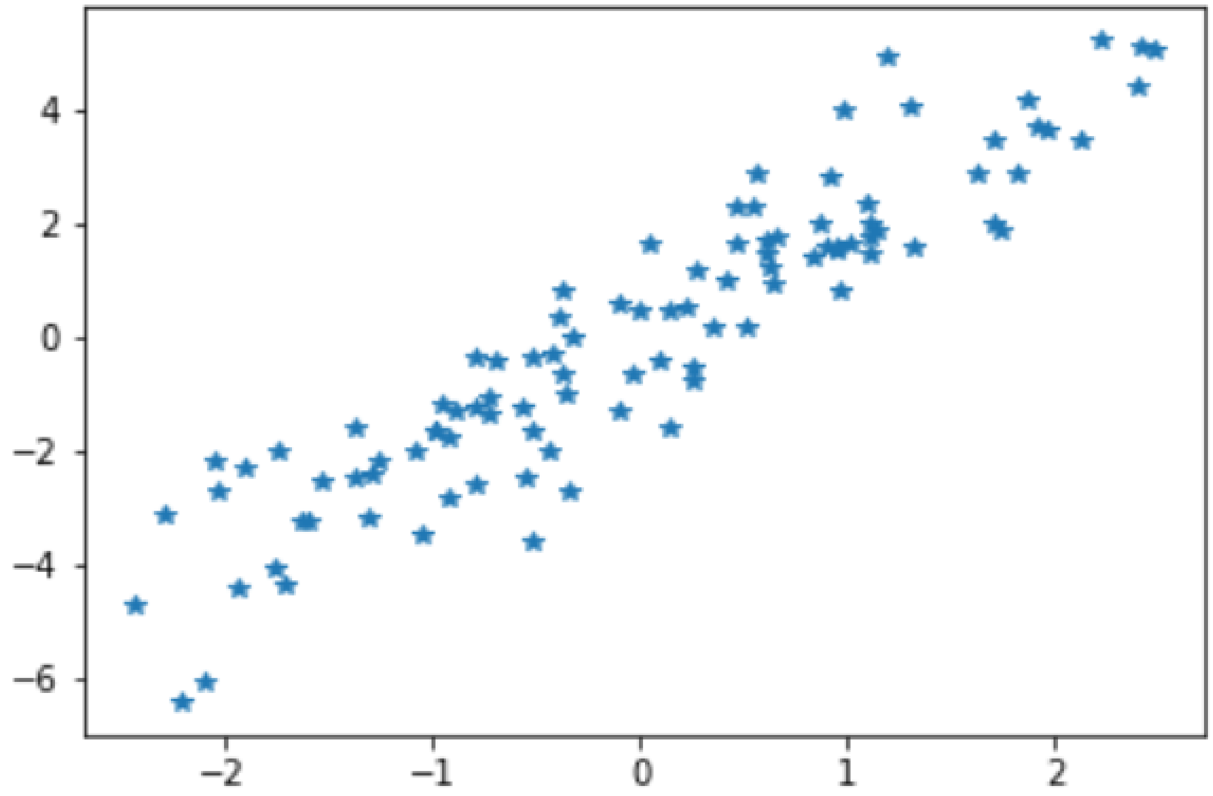
```
np.random.seed(123)
x = 5*np.random.rand(100)
y = 2*x + 1 + np.random.randn(100)

x = x.reshape(100, 1)
y = y.reshape(100, 1)

X = np.hstack([x, y]) # 水平堆叠
X.shape
```

```
(100, 2)
```

```python
%matplotlib inline
from matplotlib import pyplot as plt
plt.plot(X[:,0], X[:,1], '*')
plt.show()
```



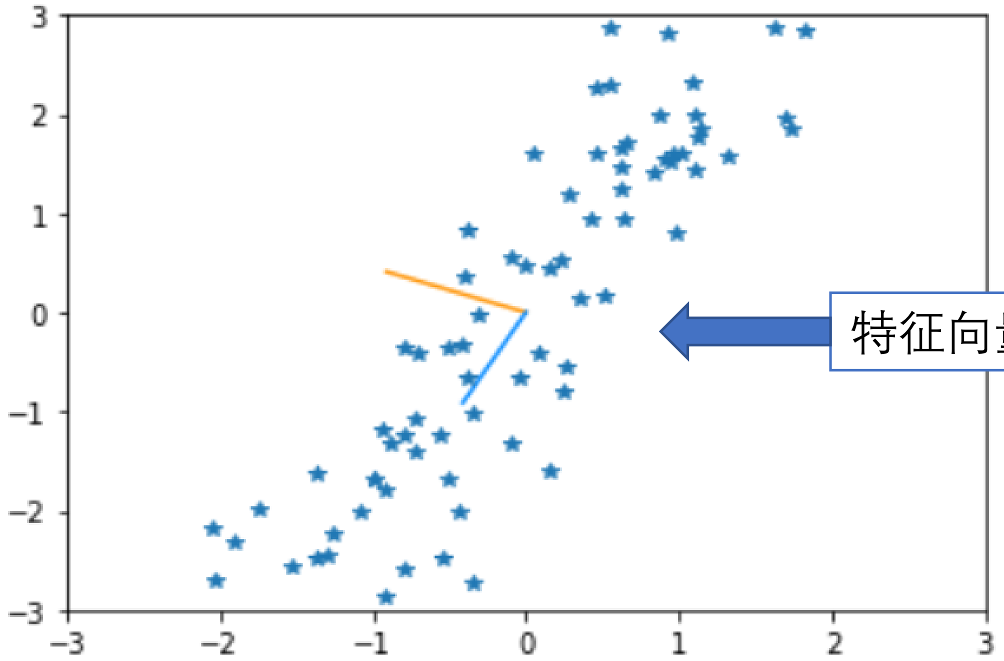
```
def centerData(X):
    X = X.copy()
    X -= np.mean(X, axis = 0)
    return X

X_centered = centerData(X)
plt.plot(X_centered[:,0], X_centered[:,1], '*')
plt.show()
```



```python
eigVals, eigVecs = np.linalg.eig(X_centered.T.dot(X_centered))
eigVecs
```

```
array([[-0.91116273, -0.41204669],
       [ 0.41204669, -0.91116273]])
```

```python
orange = '#FF9A13'
blue = '#1190FF'
plt.plot([0, eigVecs[0][0]], [0, eigVecs[1][0]], orange)
plt.plot([0, eigVecs[0][1]], [0, eigVecs[1][1]], blue)
plt.plot(X_centered[:,0], X_centered[:,1], '*')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.show()
```



特征向量、特征值的物理意义

特征降维与主成分分解 PCA

- 目标：消除特征之间的相关性，正交化的同时去掉冗余特征
- 通过空间映射变换，保留区分度大的前k个独立特征（基底向量），实现对小强度随机噪声的过滤

构造特征之间的协方差矩阵

- $C_{\vec{X}}(i;j) = Cov_{X_i,X_j} = E\left(\left(X_i - E(X_i)\right)\left(X_j - E(X_j)\right)\right)$
- 对于样本$\vec{X}^{(1)}, \dots, \vec{X}^{(N)}$,常常先让$\vec{X}^{(i)} \leftarrow \vec{X}^{(i)} - \frac{1}{N}\sum_{i=1}^{N}\vec{X}^{(i)}$ （中心化）
- 记样本矩阵为$X = \begin{pmatrix} -\vec{X}^{(1)}- \\ -\vec{X}^{(2)}- \\ \cdots \\ -\vec{X}^{(N)}- \end{pmatrix}$
- 可以估计协方差矩阵如下：

  $C(p;q) = \frac{1}{N}\sum_{k=1}^{N} X_p^{(k)} X_q^{(k)}$ （向量两两相乘）

得到属性之间的相关性

则

$C = X^T X / N$

协方差矩阵的物理意义

- $C(p;q)=\frac{1}{N}\sum_{k=1}^{N}X_p^{(k)}X_q^{(k)}$

对角线$(p;p)$上的元素：第p维特征的方差

矩阵$(p;q)$元的大小反映了所有样本第p维和第q维数据的相关性（若不相关，则为0）

PCA（主成分分解）

—— 对于实对称矩阵，存在一组正交变换使其对角化

新特征的方差

$$A = Q\Sigma Q^T = Q \begin{bmatrix} \lambda_1 & \cdots & \cdots & \cdots \\ \cdots & \lambda_2 & \cdots & \cdots \\ \cdots & \cdots & \ddots & \cdots \\ \cdots & \cdots & \cdots & \lambda_m \end{bmatrix} Q^T$$

新特征空间的基底

例子： 鸢尾花数据集的特征分析：

```
import seaborn as sns
iris = sns.load_dataset(’iris’)
print(iris.head(n = 3))
ir = iris.groupby(’species’)
ir.head(n = 2)
```

4个特征：花萼长宽，花瓣长宽

种属

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
100	6.3	3.3	6.0	2.5	virginica
101	5.8	2.7	5.1	1.9	virginica

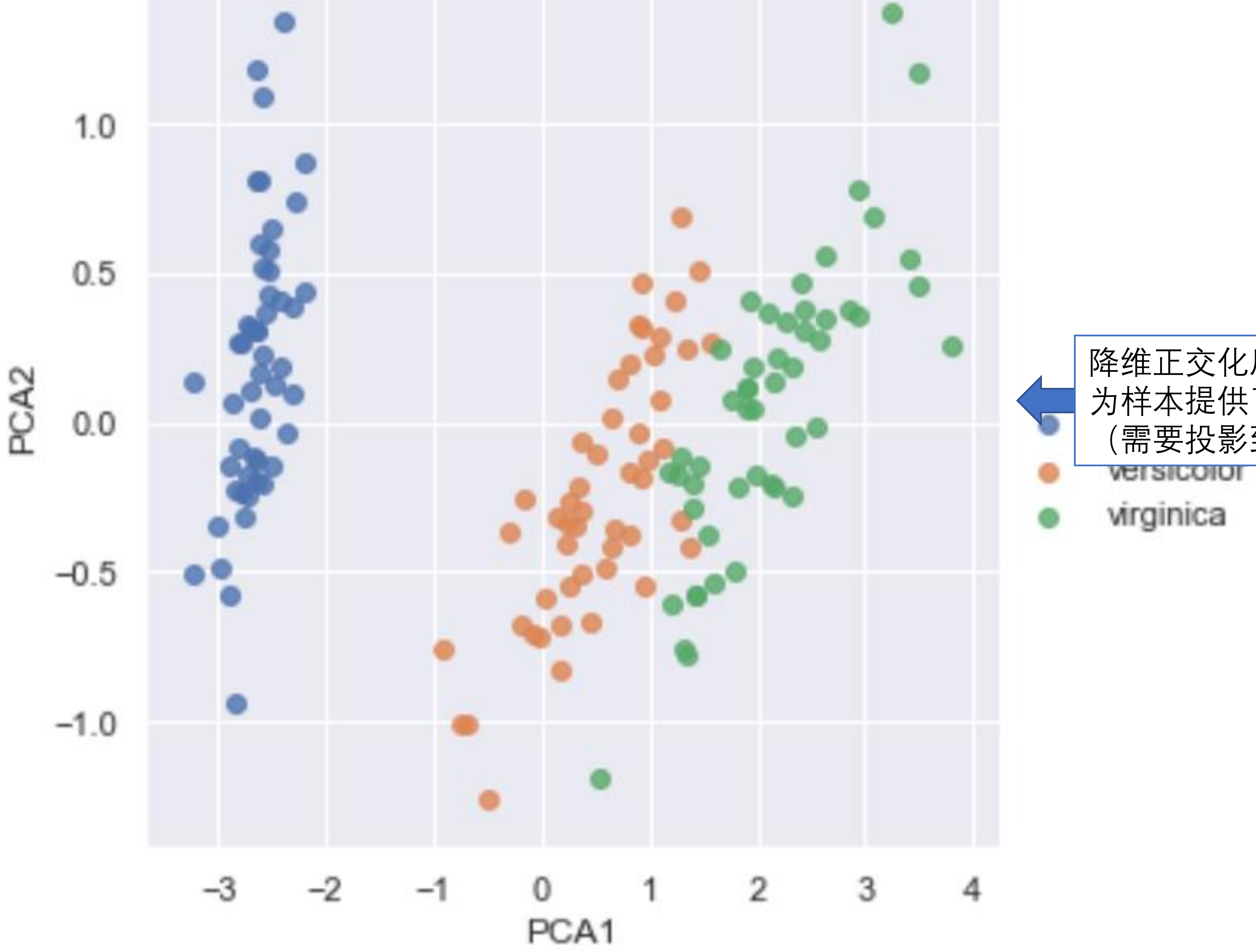
特征空间变换

- 向量乘法的物理意义是向量对于一组基的投影变换
- 变换矩阵与数据矩阵相乘，结果是将数据矩阵中的一组或多维（行）向量变换到新空间中的一组向量。目标维度与基底相同。目标向量数与原数据相同
- 一组变换矩阵（高维数组）与数据矩阵相乘，得到一组目标空间中的原数据的投影

PCA数据特征降维（数据降噪）

In [78]:
```
from sklearn.decomposition import PCA  # 1. Choose the model class
model = PCA(n_components=2)            # 2. Instantiate the model with hyperp
model.fit(X_iris)                      # 3. Fit to data. Notice y is not spec
X_2D = model.transform(X_iris)         # 4. Transform the data to two dimens
X_2D
```

Out[78]:
```
array([[-2.68412563,  0.31939725],
       [-2.71414169, -0.17700123],
       [-2.88899057, -0.14494943],
       [-2.74534286, -0.31829898],
       [-2.72871654,  0.32675451],
       [-2.28085963,  0.74133045],
       [-2.82053775, -0.08946138],
       [-2.62614497,  0.16338496],
       [-2.88638273, -0.57831175],
       [-2.6727558 , -0.11377425],
       [-2.50694709,  0.6450689 ],
       [-2.61275523,  0.01472994],
```



降维正交化后的特征空间为样本提供了更好的分布（需要投影到新空间）

Minst手写数据集的PCA-主成分分析

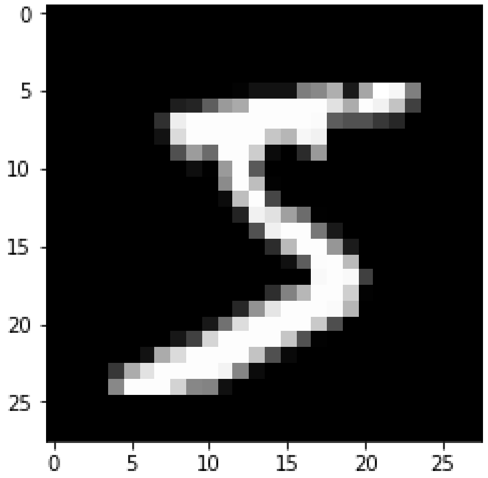
```
import pandas as pd
```

```
train = pd.read_csv('./python_course/train.csv')
print(train.shape)
```

```
(42000, 785)
```

```
target = train['label']
train = train.drop('label',axis=1)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8
0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0



PCA数据降维与聚类算法-可视化

In [27]:
```
import pandas as pd  # 读入手写体字符数据
```

In [28]:
```
train = pd.read_csv('data/train.csv')
print(train.shape)
```

```
(42000, 785)
```

In [29]:
```
target = train['label']                    # 标准答案
train = train.drop("label", axis=1)  # 样本数据
X_s = train[:6000].values     # 后面画图需要取6000个数据样本做子集
Target_s = target[:6000]
print(X.shape)
```

```
(42000, 784)
```

In [30]:
```
import numpy as np
import matplotlib.pyplot as plt

#sklearn包里的PCA，用的是机器学习的最小loss拟合方案
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

用Numpy实现PCA

```
X = train.values
X.shape
```

```
(42000, 784)
```

```
X_std = StandardScaler().fit_transform(X)
X_std_s = StandardScaler().fit_transform(X_s)

# Calculating Eigenvectors and eigenvalues of Cov matirx
mean_vec = np.mean(X_std, axis=0)                    # 中心化、标准化(归一化)
cov_mat = np.cov(X_std.T)                            # 协方差矩阵
eig_vals, eig_vecs = np.linalg.eig(cov_mat)          # 求出特征值特征向量 Numpy版
# Create a list of (eigenvalue, eigenvector) tuples
eig_pairs = [ (np.abs(eig_vals[i]),eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the eigenvalue, eigenvector pair from high to low
eig_pairs.sort(key = lambda x: x[0], reverse= True)

# Calculation of Explained Variance from the eigenvalues
tot = sum(eig_vals)
var_exp = [(i/tot)*100 for i in sorted(eig_vals, reverse=True)]  # 求出特征值的总能量占比
cum_var_exp = np.cumsum(var_exp) # Cumulative explained variance   #积分累加
```

标准化，保证每个维度的数据方差是1，均值为0，并不是所有任务都适用

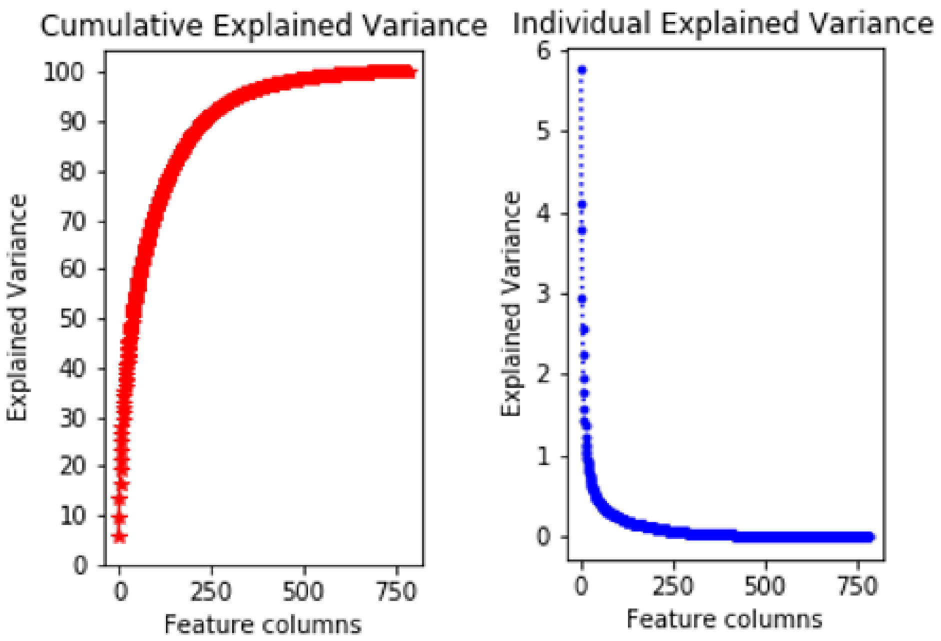
PCA-主成分分析

```
import matplotlib.pyplot as plt
```

```
fig = plt.figure(figsize=(6,8))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
plt.subplots_adjust(wspace=0.5, hspace=0)
x = list(range(784))
ax1.plot(x, cum_var_exp, color='r', linestyle='-', marker='*')
ax2.plot(x, var_exp, color='b', linestyle=':', marker='.')
ax1.set_title('Cumulative Explained Variance')
ax1.set_xlabel('Feature columns')
ax1.set_ylabel('Explained Variance')
ax1.set_yticks(list(range(0,101,10)))
props = {
    'title': 'Individual Explained Variance',
    'xlabel': 'Feature columns',
    'ylabel': 'Explained Variance'
}
ax2.set(**props)
```

建立两个子图

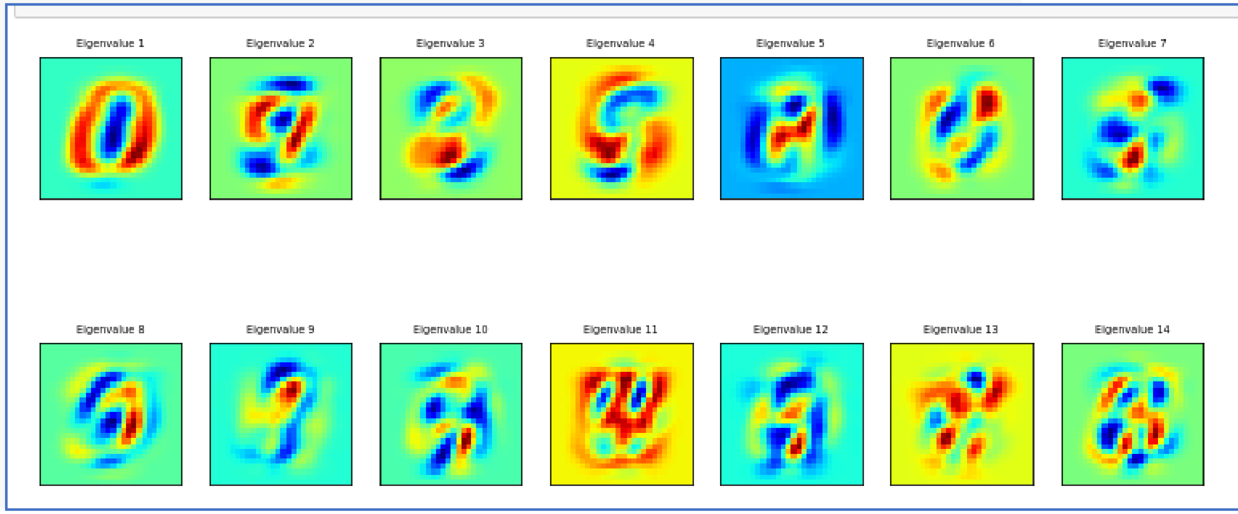
进行绘制，设置坐标轴，图名，刻度范围等



```
# Invoke SKlearn's PCA method
n_components = 30
pca = PCA(n_components=n_components).fit(X)  # 拟合最小损失的30维PCA空间，SKlearn版

eigenvectors = pca.components_.reshape(n_components, 28, 28) # 把特征向量转成二维图片

# Extracting the PCA components ( eignevalues )
eigenvalues = pca.singular_values_
eigenvalues.shape
```

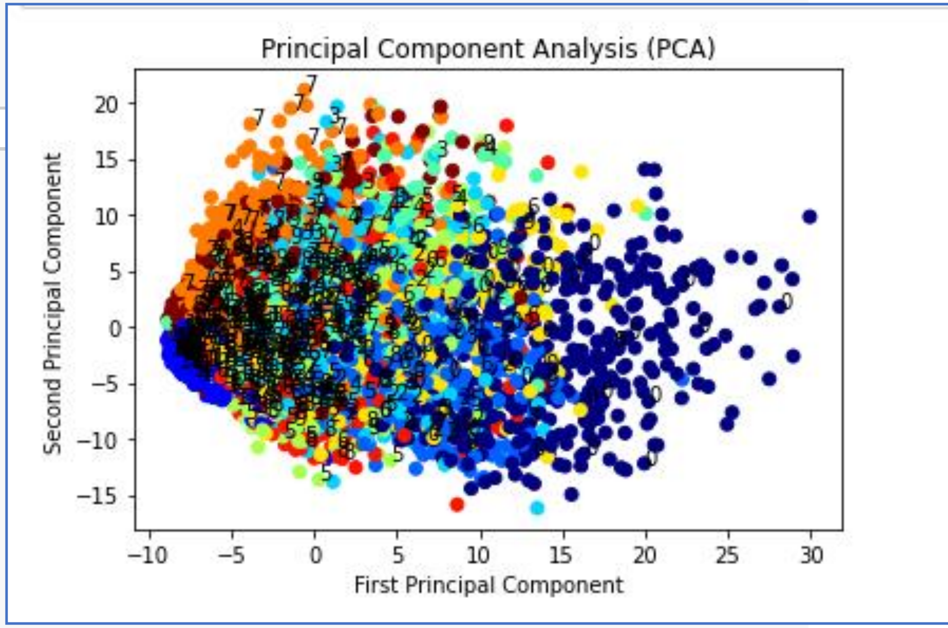
```
(30,)
```

```
# Plot the first 2*7 eignenvectors
n_row = 2
n_col = 7

plt.figure(figsize=(13,6))   # 画框大小
for i in list(range(n_row * n_col)):
    offset = 0
    plt.subplot(n_row, n_col, i + 1) # 行列标定子图
    plt.imshow(eigenvectors[i].reshape(28,28), cmap='jet')
    title_text = 'Eigenvalue ' + str(i + 1)
    plt.title(title_text, size=6.5)
    plt.xticks(())
    plt.yticks(())
plt.show()
```

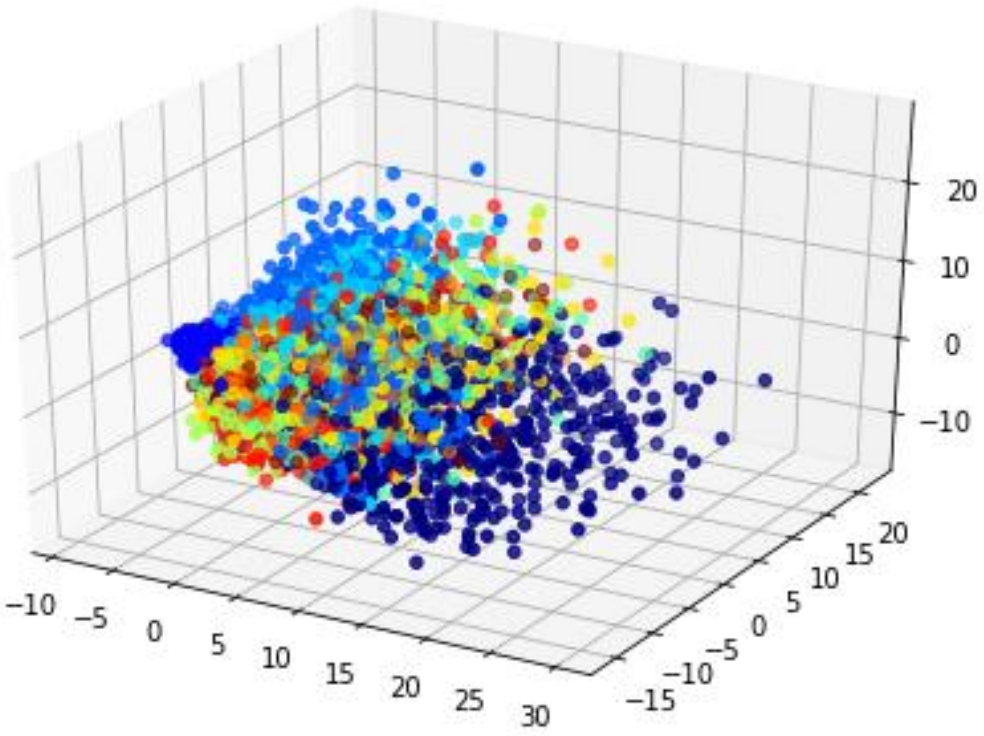


```python
pca = PCA(n_components = 3)
pca.fit(X_std)    # 生成3维基底的特征空间

# 把原空间6000个子集的数据784维，投影到新3D特征空间中
# 这一步非常重要，PCA只是学到了一个基底
# 具体数据效果还是要把数据投影进去，可以思考Numpy下该如何实现？
X_3d = pca.transform(X_std_s)
# X_3d[1]
```

```python
fig = plt.figure()
ax1 = fig.add_subplot(111)
#设置标题
ax1.set_title('Principal Component Analysis (PCA)')
#设置X轴标签
plt.xlabel('First Principal Component')
#设置Y轴标签
plt.ylabel('Second Principal Component')
# 这里只按前两维画散点，不同target颜色不同
ax1.scatter(X_3d[:,0],X_3d[:,1],c = Target, cmap='jet', marker = 'o')

for i in range(0,6000,10):    # 为了能看清楚，这里用了小样本的6000个数据
    ax1.annotate(str(Target[i]), (X_3d[i, 0], X_3d[i, 1])) # 根据实际标签显示不同数字
#设置图标
#显示所画的图
plt.show()
```



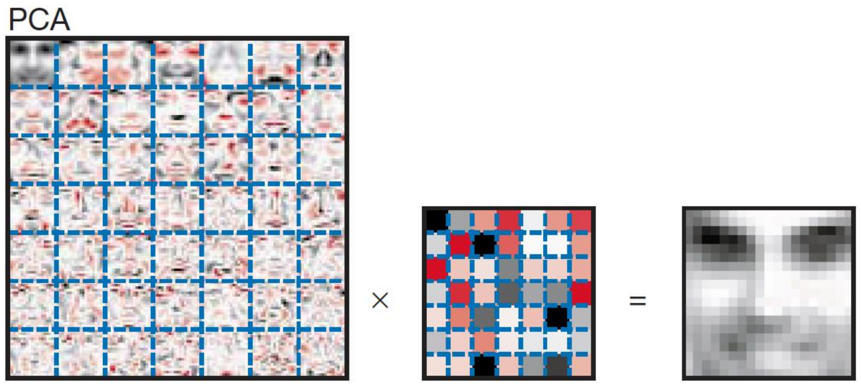
```
pca = PCA(n_components=3)
pca.fit(X_std)
X_3d = pca.transform(X_std)


from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)    # 3维散点图

ax.scatter(X_3d[:, 0], X_3d[:, 1], X_3d[:, 2], c=Target,  cmap='jet', marker = 'o')
plt.show()
```



PCA的物理直观：

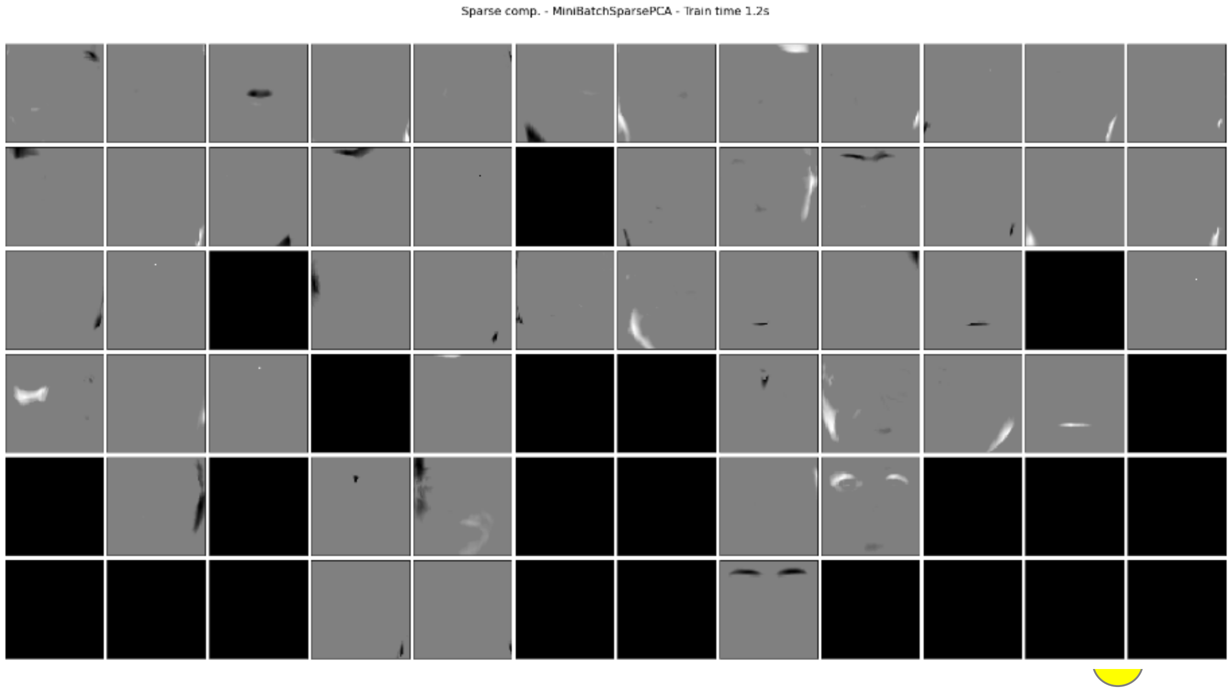
- PCA的目标：降维的同时保留大部分信息
  - 直观地理解：数据投影之后的值在特征轴上尽可能分散
  - 同时我们希望各个维度尽可能不相关
    - 因此自然引入协方差矩阵：方差尽可能大（协方差矩阵的对角线）
    - 常用于同类型样本数集的降维

人脸数据集的特征降维


PCA
×
=


First centered Olivetti faces


Eigenfaces - PCA using randomized SVD - Train time 0.0s


Sparse comp. - MiniBatchSparsePCA - Train time 1.2s

HITS算法：基于超文本链接的主题搜索排名

关键词检索生成base-set

通过base-set网页间的超链接关系生成扩充candidate –set

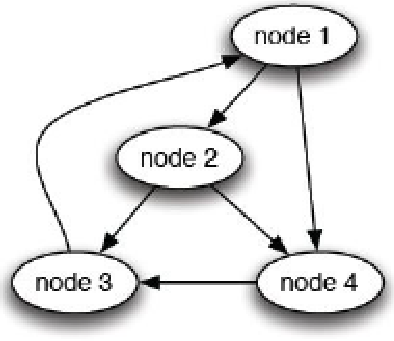
Items之间的关联构成相互推荐的一个网络（矩阵）

HITS算法求解最佳排名

推荐问题与奇异向量

A	甲	乙	丙	丁	1st得分（hub值）	2nd得分
家园	★		★	★	3	21
艺园	★	★	★		3	20
康博思		★			1	6
松林	★			★	2	15
燕南		★		★	2	13
权威值	1	1	1	1		
权威值	8	6	6	7		

网络分析基本思想
静态结构分析：基础统计
基础统计
社会群体及社会角色
弱关系及桥
点的同质性(homophily)：强关系的影响
对强弱关系的反思
对结构的深入研究：建立模型
网络平衡
度数分布的深入研究：模型的实例
网络的链接分析及预测
数据收集及处理
抽样方法
作图



$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 6 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 9 \\ 7 \\ 2 \\ 4 \end{bmatrix}$$

定义n个网页之间链接关系的邻接矩阵为M,即M_{ij}为1$\Longleftrightarrow$从网页i到j有链接，则网页i的中枢值为：

$$a_i \longleftarrow M_{1i}h_1 + M_{2i}h_2 + \ldots M_{ni}h_n \tag{3}$$

$$h_i \longleftarrow M_{i1}a_1 + M_{i2}a_2 + \ldots M_{in}a_n \tag{4}$$

h^k=$(h_1, h_2, \ldots h_n)^T$,a^k=$(a_1, a_2, \ldots a_n)^T$为运行了k次的时候，n个网页的中枢，权威向量，则转换规则为：（先更新a^k)

$$a^k = M^T h^{k-1} \tag{5}$$

$$h^k = Ma^k \tag{6}$$

h^0, a^0的元素都为1 迭代可得：$a^1 = M^T h^0, h^1 = MM^T h^0, \ldots$

$$a^k = (M^T M)^{k-1} h^0 \tag{7}$$

$$h^k = (MM^T)^k h^0 \tag{8}$$

定理：n*n的实对称矩阵有n个特征值，且不同特征值的特征向量彼此正交（即特征向量构成线性空间的一组基底）

设 MM^T 的特征值为$c_1,c_2,\ldots c_n$,且$c_1>c_2>,\ldots>c_n$,对应的特征向量为$z_1,z_2,\ldots z_n$,而h^0在基底下的表示为

$$h_0 = q_1z_1 + q_2z_2 + \ldots + q_nz_n \tag{9}$$

则

$$\begin{aligned}
h^k &= (MM^T)^k h^0 && (10)\\
&= (MM^T)^k(q_1z_1 + q_2z_2 + \ldots q_nz_n) && (11)\\
&= q_1(MM^T)^k z_1 + q_2(MM^T)^k z_2 + \ldots + q_n(MM^T)^k z_n && (12)\\
&= q_1c_1^k z_1 + q_2c_2^k z_2 + \ldots q_nc_n^k z_n && (13)
\end{aligned}$$

如果要收敛，需要对每项正规化。则

$$h^k = = \frac{(MM^T)^k h^0}{\|(MM^T)^k h^0\|} \tag{14}$$

$$= \frac{q_1 c_1^k z_1 + q_2 c_2^k z_2 + \ldots q_n c_n^k z_n}{\|q_1 c_1^k z_1 + q_2 c_2^k z_2 + \ldots q_n c_n^k z_n\|} \tag{15}$$

$$= \frac{q_1 z_1 + q_2 (\frac{c_2}{c_1})^k z_2 + \ldots q_n (\frac{c_2}{c_1})^k z_n}{\|q_1 z_1 + q_2 (\frac{c_2}{c_1})^k z_2 + \ldots q_n (\frac{c_2}{c_1})^k z_n\|} \tag{16}$$

$$= \frac{q_1 z_1}{\|q_1 z_1\|} \tag{17}$$

故h_k收敛，同理可得a_k收敛

奇异值分解（SVD）与 特征分解

- 奇异值分解的数学方案
- 特征分解的物理本质
- 隐含语义挖掘（LSI 或称 LSA）

奇异值分解的数学表达

- $N \times M$矩阵X，把每一行当成一个点。设$r = rank(X)$。
- $X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \dots & \vec{u}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} -\vec{v}_1^T - \\ \dots \\ -\vec{v}_r^T \end{bmatrix} = UDV^T$
- U, D, V分别为$N \times r, r \times r, M \times r$矩阵
  - 特征空间变换
- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$称为奇异值
  - 其中：$\vec{u}_i \vec{v}_i^T$ 要满足归一化和正交化的要求

奇异值分解的基本求解流程

- $N \times M$矩阵X，把每一行当成一个点。设$r = rank(X)$。
- $\vec{v}_1 = argmax_{|\vec{v}|=1}|X\vec{v}|$
- $\vec{v}_2 = argmax_{|\vec{v}|=1,\vec{v}\perp\vec{v}_1}|X\vec{v}|$ ⟵ 保证正交的前提下的下一个可最大拉伸的方向
- ……
- $\vec{v}_r = argmax_{|\vec{v}|=1,\vec{v}\perp\vec{v}_1,\vec{v}\perp\vec{v}_2\dots,\vec{v}\perp\vec{v}_{r-1}}|X\vec{v}|$

- 令$\sigma_i = |X\vec{v}_i|, \overrightarrow{u_i} = \frac{1}{\sigma_i}X\vec{v}_i, 1 \le i \le r$
- 则$X = \sum_{i=1}^{r}\sigma_i\vec{u}_i\vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \dots & \vec{u}_r \\ | & & | \end{bmatrix}\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}\begin{bmatrix} -\vec{v}_1^T - \\ \dots \\ -\vec{v}_r^T \end{bmatrix} = UDV^T$

奇异值分解与降维：泛化与泛化损失

- $X = UDV^T$
- 按方差大小排列，得到第1,2,⋯, r个奇异值
- 奇异值小的几个主成分可以认为是不显著特征，将其丢弃——降维
- 则$X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \dots & \vec{u}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} -\vec{v}_1^T - \\ \dots \\ -\vec{v}_r^T \end{bmatrix} = UDV^T$

SVD图像降维的例子：Numpy Tutorials

```
from scipy import misc

img = misc.face()
print(type(img), img.shape)
```

```
<class 'numpy.ndarray'> (768, 1024, 3)
```

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.imshow(img)
plt.show()
```

```python
img_array = img / 255  # 色彩值（灰度值） 归一化
red_array = img_array[:, :, 0]
green_array = img_array[:, :, 1]
blue_array = img_array[:, :, 2]
img_gray = img_array @ [0.2126, 0.7152, 0.0722]  # 0.2126R, 0.

#plt.imshow(blue_array*255)  # RGB values (0-1 float or 0-255
#plt.show()
plt.imshow(img_gray, cmap="gray")
plt.show()
```

```python
from numpy import linalg
U, s, Vt = linalg.svd(img_gray)  # When a is a 2D array, it is
U.shape, s.shape, Vt.shape       # s只有一维，奇异值向量
```

((768, 768), (768,), (1024, 1024))

```python
import numpy as np

Sigma = np.zeros((U.shape[1], Vt.shape[0]))
np.fill_diagonal(Sigma, s)  # 补全s成为对角矩阵
```

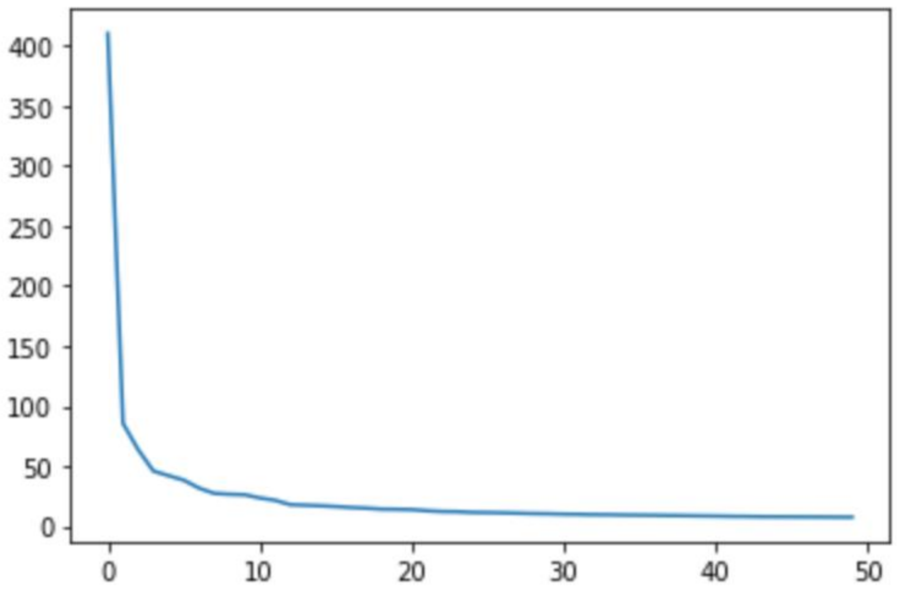
```python
linalg.norm(img_gray - U @ Sigma @ Vt)  # 计算还原误差
```

1.3552105737617506e-12

```python
np.allclose(img_gray, U @ Sigma @ Vt) # Returns True if two arr
```

True

```
plt.plot(s[:50])  # 观察一下前50个奇异值的取值分布情况（能量占l
plt.show()
```



```
k = 20
approx = U @ Sigma[:, :k] @ Vt[:k, :]
plt.imshow(approx, cmap="gray")
plt.show()
```

np.transpose(x, axes=(i, j, k)) indicates that the axis will be reordered such that the final shape of the transposed array will be reordered according to the indices (i, j, k).

```
img_array_transposed = np.transpose(img_array, (2, 0, 1))
img_array_transposed.shape
```

(3, 768, 1024)

```
U, s, Vt = linalg.svd(img_array_transposed)  # 三通道色彩矩阵的
U.shape, s.shape, Vt.shape
```

((3, 768, 768), (3, 768), (3, 1024, 1024))

```
Sigma = np.zeros((3, 768, 1024))
for j in range(3):
    np.fill_diagonal(Sigma[j, :, :], s[j, :]) # 生成3通道sigma

reconstructed = U @ Sigma @ Vt
reconstructed.shape
```

(3, 768, 1024)

```
reconstructed.min(), reconstructed.max()
```

(-6.305656250670695e-15, 1.000000000000004)

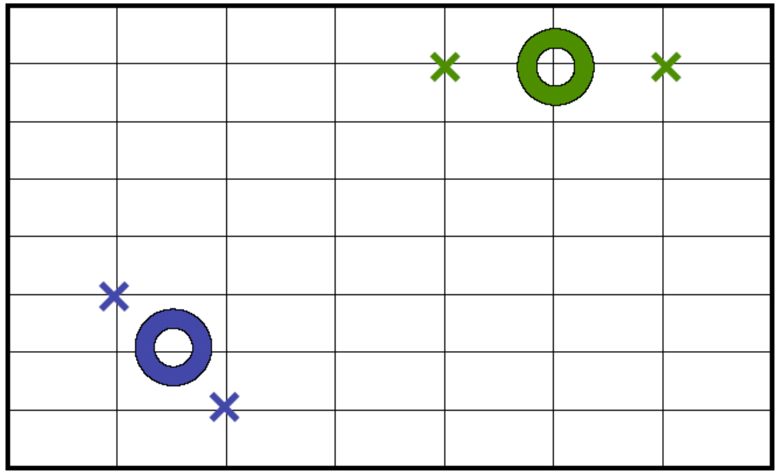
```
reconstructed = np.clip(reconstructed, 0, 1)          # 裁剪掉负
plt.imshow(np.transpose(reconstructed, (1, 2, 0)))    # 恢复原形
plt.show()
```

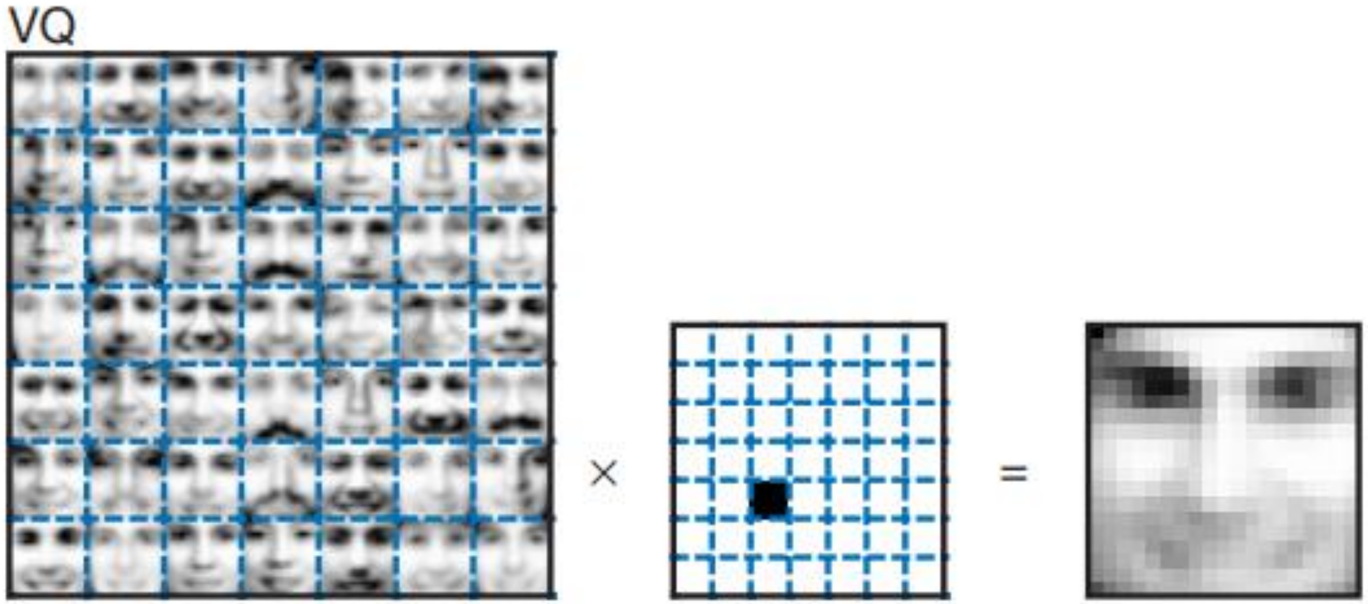
```
approx_img = U @ Sigma[..., :k] @ Vt[..., :k, :]
plt.imshow(np.clip(np.transpose(approx_img, (1, 2, 0)),0,1))
plt.show()
```

Latent Semantic Indexing (LSI)

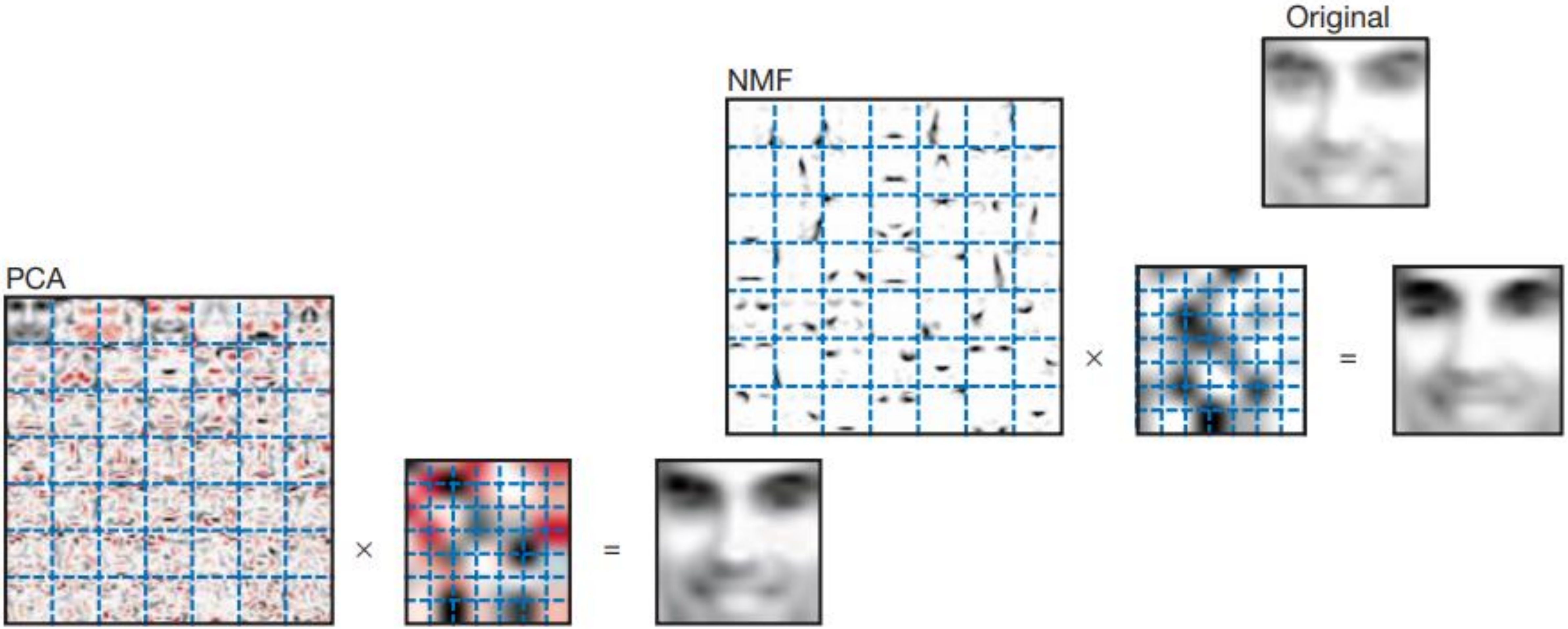
$$
\begin{array}{l}
\text{Terms} \\
\text{a} \\
\text{arrived} \\
\text{damaged} \\
\text{delivery} \\
\text{fire} \\
\text{gold} \\
\text{in} \\
\text{of} \\
\text{shipment} \\
\text{silver} \\
\text{truck}
\end{array}
\qquad
\mathbf{A} = \begin{array}{ccc}
d1 & d2 & d3
\end{array}
\begin{bmatrix}
1 & 1 & 1 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 0 & 1 \\
0 & 2 & 0 \\
0 & 1 & 1
\end{bmatrix}
\qquad
\mathbf{q} = \begin{array}{c} q \end{array}
\begin{bmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1
\end{bmatrix}
$$

K-means聚类与VQ（向量 量化）





NMF特征分解 与 PCA



https://www.nature.com/articles/44565.pdf

特征分解与主成分分析的应用场景

- 都会要求产生一组正交基
- 主成分分析：把样本投影到新的特征空间
  - 特征工程（检索、聚类）、数据降噪、数据压缩
- 特征分解（SDV）：样本集被分解为 参数*特征 的线性表达
  - 文档话题模型（主题分析）
  - 隐含语义挖掘（词义泛化、协同过滤）：降维后乘回矩阵X'
  - 检索、聚类（也需要把样本投到新的特征空间）
  - 特征分析与压缩（GIF、jpg）：可以放松正交性约束

Numpy部分内容总结

- Numpy提供了一个面向矩阵访问，矩阵算术计算，矩阵向量计算及广播机制的软件平台
  - 与C语言的函数街廓和数据结构兼容性保证了其执行效率
  - 附带的科学计算函数集为相关应用提供了方便
  - 矩阵运算也是当前神经网络计算的基础

- 矩阵分解本质上可以认为是一种反向的模型参数计算方案。在给定损失目标的情况下，返回最优解或可行解。通常可以由一组矩阵计算的流程迭代实现

距离与核函数

- 欧氏距离:
- 马氏距离:

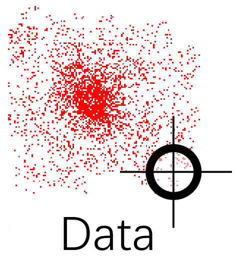
$$D_M(x,y)=\sqrt{(x-y)^T\Sigma^{-1}(x-y)}$$

- 核函数加权估计

距离相关的核函数 Various Kernels

$$P(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K(\mathbf{x}-\mathbf{x}_i)$$

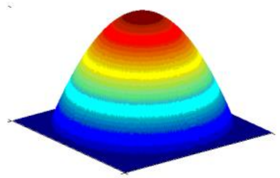
A function of some finite number of data points
$x_1 \cdots x_n$



Examples:

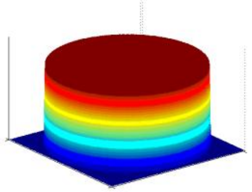
- Epanechnikov Kernel

$$K_E(\mathbf{x}) = \begin{cases} c\left(1-\|\mathbf{x}\|^2\right) & \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$



- Uniform Kernel

$$K_U(\mathbf{x}) = \begin{cases} c & \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$



- Normal Kernel

$$K_N(\mathbf{x}) = c \cdot \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right)$$