


Pandas基础 (C05)



2020/03/17

北京大学信息科学技术学院

主要内容

- ➡ Numpy (补充)
- ➡ Pandas 基础
- ➡ 数据特征分析 PCA
- ➡ 网络通讯基础 (梁钧鋆助教)

Universal functions (ufunc)

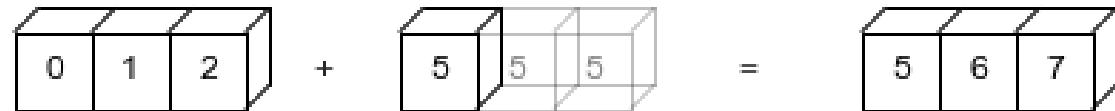
A universal function (or **ufunc** for short) is a function that operates on **ndarrays** in an element-by-element fashion, supporting **array broadcasting**, **type casting**, and several other standard features. That is, a ufunc is a “**vectorized**” wrapper for a function that takes a fixed number of specific inputs and produces a fixed number of specific outputs.

In NumPy, universal functions are instances of the **numpy.ufunc** class. Many of the built-in functions are implemented in compiled C code. The basic ufuncs operate on scalars, but there is also a generalized kind for which the basic elements are sub-arrays (vectors, matrices, etc.), and broadcasting is done over other dimensions. One can also produce custom **ufunc** instances using the **frompyfunc** factory function.

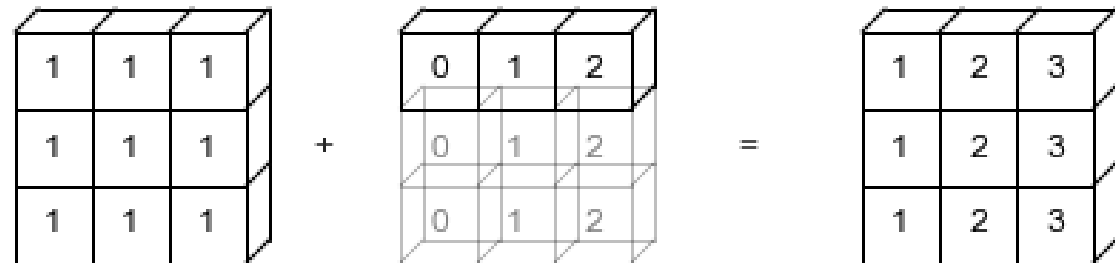
Broadcasting

Each universal function takes array inputs and produces array outputs by performing the core function element-wise on the inputs (where an element is generally a scalar, but can be a vector or higher-order sub-array for generalized ufuncs). Standard broadcasting rules are applied so that inputs not sharing exactly the same shapes can still be usefully operated on.

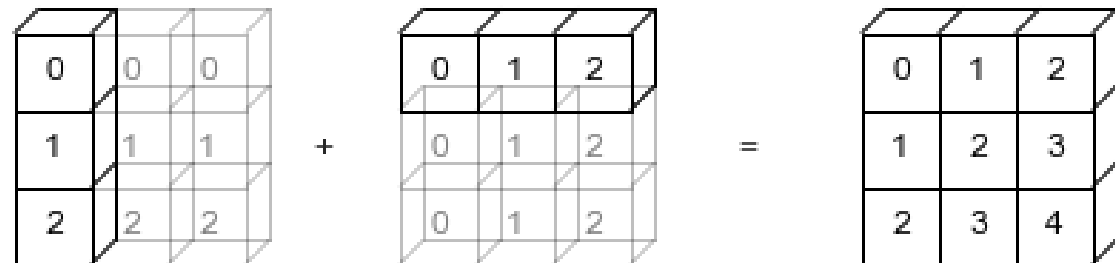
`np.arange(3) + 5`



`np.ones((3, 3)) + np.arange(3)`



`np.arange(3).reshape((3, 1)) + np.arange(3)`



```
import numpy as np
np.random.seed(0)

def compute_reciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1.0 / values[i] # 求倒数
    return output

values = np.random.randint(1, 10, size=5)
compute_reciprocals(values)
```

```
array([0.16666667, 1.          , 0.25         , 0.25         , 0.125        ])
```

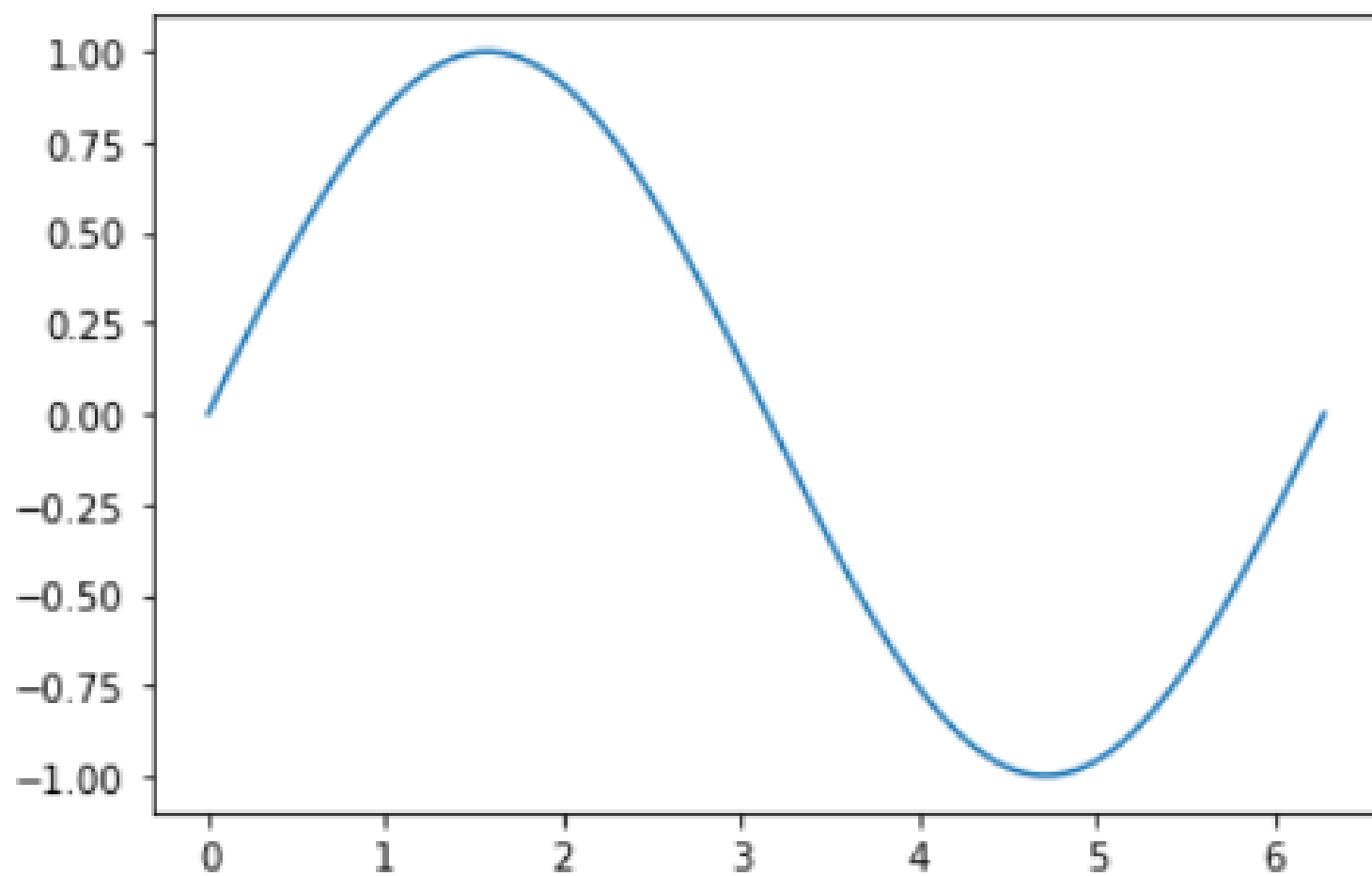
```
1.0 / values # 效率高3倍
```

```
array([0.16666667, 1.          , 0.25         , 0.25         , 0.125        ])
```

```
theta = np.linspace(0, np.pi * 2, 100)
y = np.sin(theta) # 单目函数 广播

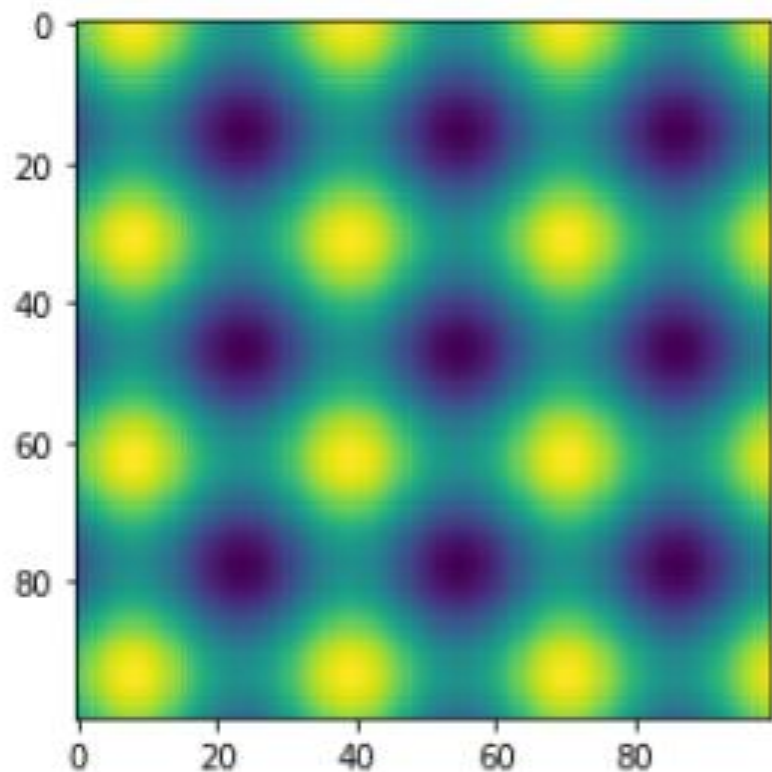
plt.plot(theta, y)
```

Out[4]: [<matplotlib.lines.Line2D at 0x23bc9911fc8>]



```
: # define a function  $z = f(x, y)$   
#  $x$  and  $y$  have 100 steps from 0 to 10  
  
x = np.linspace(0, 10, 100)  
y = np.linspace(0, 10, 100)[:, np.newaxis] # 增加一个维度  
  
z = np.sin(2*x) + np.cos(2*y) # 广播合成2维度矩阵  
plt.imshow(z)
```

```
: <matplotlib.image.AxesImage at 0x23bca086c88>
```



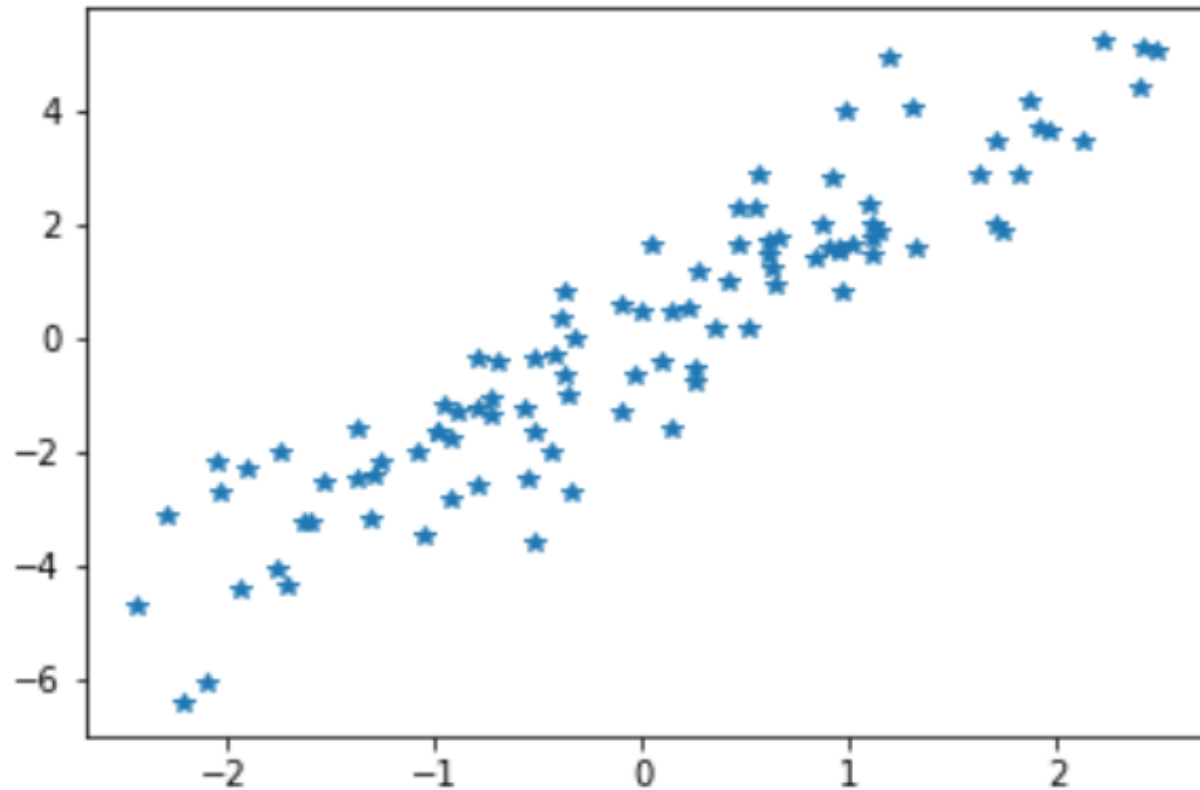
```
x = np.arange(1, 6)  
np.multiply.outer(x, x)
```

```
array([[ 1,  2,  3,  4,  5],  
       [ 2,  4,  6,  8, 10],  
       [ 3,  6,  9, 12, 15],  
       [ 4,  8, 12, 16, 20],  
       [ 5, 10, 15, 20, 25]])
```

Aggregation functions (聚合函数)

Function Name	NaN-safe Version	Description
<code>np. sum</code>	<code>np. nansum</code>	Compute sum of elements
<code>np. prod</code>	<code>np. nanprod</code>	Compute product of elements
<code>np. mean</code>	<code>np. nanmean</code>	Compute mean of elements
<code>np. std</code>	<code>np. nanstd</code>	Compute standard deviation
<code>np. var</code>	<code>np. nanvar</code>	Compute variance
<code>np. min</code>	<code>np. nanmin</code>	Find minimum value
<code>np. max</code>	<code>np. nanmax</code>	Find maximum value
<code>np. argmin</code>	<code>np. nanargmin</code>	Find index of minimum value
<code>np. argmax</code>	<code>np. nanargmax</code>	Find index of maximum value
<code>np. median</code>	<code>np. nanmedian</code>	Compute median of elements
<code>np. percentile</code>	<code>np. nanpercentile</code>	Compute rank-based statistics of elements


```
1 def centerData(X):  
2     X = X.copy()  
3     X -= np.mean(X, axis = 0)  
4     return X  
5  
6 X_centered = centerData(X)  
7 plt.plot(X_centered[:,0], X_centered[:,1], '*')  
8 plt.show()
```





Pandas — Panel data analysis

- 序列: indexed list
- 多通道序列: record list
- 多字段二维表

The Pandas Series Object —— 序列

A Pandas `Series` is a one-dimensional array of indexed data. It can be created from a list or array as follows:

缺省情况类似excel的表格，自动维护标号索引

```
1 data = pd.Series([0.25, 0.5, 0.75, 1.0])
2 print(data)
3 data.index
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

与数组类似，支持下标访问操作

```
1 data.values
```

```
array([ 0.25,  0.5 ,  0.75,  1.  ])
```

The index is an array-like object of type `pd. Index`

```
1 data[1]
```

```
0.5
```

```
1 data[1:3]
```

```
1    0.50
```

```
2    0.75
```

```
dtype: float64
```

也可以指定可哈希的索引项，类似dict

```
1 data = pd.Series([0.5, 0.25, 1.75, 1.0],  
2                  index=['a', 'b', 'c', 'd']) ←  
3 print(data)  
4 print(data.sort_values())  
5 data['b'] ←
```

```
a    0.50  
b    0.25  
c    1.75  
d    1.00  
dtype: float64  
b    0.25  
a    0.50  
d    1.00  
c    1.75  
dtype: float64
```

0.25

We can even use non-contiguous or non-sequential indices:

```
1 data = pd.Series([0.25, 0.5, 0.75, 1.0],
2                   index=[2, 5, 3, 7]) ←
3 data[5]
```


0.5

```
1 data[data>0.7] * 2
```

```
3    1.5 ←
```

```
7    2.0
```

```
dtype: float64
```



```
1 print(0.75 in data)
2 0.75 in data.values
```

False

True

```
1 for i in data.values: ←
2     print(i)
```


0.25

0.5

0.75

1.0

True



```
1 a = pd.Series([2, 4, 6])
2 b = pd.Series({2:'a', 1:'b', 3:'c'})
3 print(b[1])
4 2 in b
```

b

True

```
1 for i in b:
2     print (i)
```

a

b

c


```
1 sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
2 obj3 = pd.Series(sdata)
3 print(obj3)
4 states = ['California', 'Ohio', 'Oregon', 'Texas']
5 obj4 = pd.Series(sdata, index=states) ← 插入索引
6 obj4
```

```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

```
California    NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

由于"California"所对应的sdata值找不到，所以其结果就为NaN（即“非数字”（not a number），在pandas中，它用于表示缺失或NA值）。因为‘Utah’不在states中，它被从结果中除去。

```
1 # Series最重要的一个功能是，它会根据运算的索引标签自动对齐数据
2 # 关于数据对齐功能如果你使用过数据库，可以认为是类似join的操作
3 obj3+obj4
```

```
California      NaN
Ohio            70000.0
Oregon          32000.0
Texas           142000.0
Utah            NaN
dtype: float64
```

```
1 obj3 - obj4
```

```
California      NaN
Ohio            0.0
Oregon          0.0
Texas           0.0
Utah            NaN
dtype: float64
```

The Pandas DataFrame Object

- 视角1：多个对齐的序列（series）的组合
- 视角2：多维度的 Numpy array 支持 多维度索引
- 视角3：多帧数据的序列，每帧数据是一个Numpy array

索引-数据 与 索引合并

```
1 population_dict = {'California': 38332521,
2                    'Texas': 26448193,
3                    'New York': 19651127,
4                    'Florida': 19552860,
5                    'Illinois': 12882135}
6 population = pd.Series(population_dict)
7
8 area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,
9             'Florida': 170312, 'Illinois': 149995}
10 area = pd.Series(area_dict)
```

```
1 states = pd.DataFrame({'population': population, 'area': area})
2 states
```

	population	area
California	38332521	423967
Texas	26448193	695662
New York	19651127	141297
Florida	19552860	170312
Illinois	12882135	149995

```

1 population_dict = {'California': 38332521, 'Texas': 26448193,
2                    'New York': 19651127, 'W.DC': 11000000}
3 population = pd.Series(population_dict)
4
5 area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,
6             'Florida': 170312, 'Illinois': 149995}
7 area = pd.Series(area_dict)

```

```

1 states = pd.DataFrame({'population': population, 'area': area})
2 states

```

	population	area
California	38332521.0	423967.0
Florida	NaN	170312.0
Illinois	NaN	149995.0
New York	19651127.0	141297.0
Texas	26448193.0	695662.0
W.DC	11000000.0	NaN

索引-数据 与 索引合并 (续)

多重索引

```
1 print(states.index)
2 print(states.columns) ←
3 for i in states.columns:
4     print(states[i])
```

Index(['California', 'Florida', 'Illinois', 'New York', 'Texas', 'W.DC'], dtype='object')

Index(['population', 'area'], dtype='object') ←

California 38332521.0

Florida NaN

Illinois NaN

New York 19651127.0

Texas 26448193.0

W.DC 11000000.0

Name: population, dtype: float64

California 423967.0

Florida 170312.0

Illinois 149995.0

New York 141297.0

Texas 695662.0

W.DC NaN

Name: area, dtype: float64

表5-1：可以输入给DataFrame构造器的数据

类型	说明
二维ndarray	数据矩阵，还可以传入行标和列标
由数组、列表或元组组成的字典	每个序列会变成DataFrame的一列。所有序列的长度必须相同
NumPy的结构化/记录数组	类似于“由数组组成的字典”
由Series组成的字典	每个Series会成为一列。如果没有显式指定索引，则各Series的索引会被合并成结果的行索引
由字典组成的字典	各内层字典会成为一列。键会被合并成结果的行索引，跟“由Series组成的字典”的情况一样
字典或Series的列表	各项将会成为DataFrame的一行。字典键或Series索引的并集将会成为DataFrame的列标
由列表或元组组成的列表	类似于“二维ndarray”
另一个DataFrame	该DataFrame的索引将会被沿用，除非显式指定了其他索引
NumPy的MaskedArray	类似于“二维ndarray”的情况，只是掩码值在结果DataFrame会变成NA/缺失值

词典的列表生成dataframe:

If some keys in the dictionary are missing, Pandas will fill them in with `NaN` (i.e., "not a number") values:

```
: 1 data = [{'a': i, 'b': 2 * i}
    2         for i in range(3)]
    3 pd.DataFrame(data)
```

	a	b
0	0	0
1	1	2
2	2	4

```
: 1 pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}])
```

	a	b	c
0	1.0	2	NaN
1	NaN	3	4.0

From a two-dimensional NumPy array

Given a two-dimensional array of data, we can create a `DataFrame` with any specified column and index names. If omitted, an integer index will be used for each:

```
1 pd.DataFrame(np.random.rand(3, 2),  
2               columns=['foo', 'bar'],  
3               index=['a', 'b', 'c'])
```

	foo	bar
a	0.865257	0.213169
b	0.442759	0.108267
c	0.047110	0.905718

数据特征与主成分分解

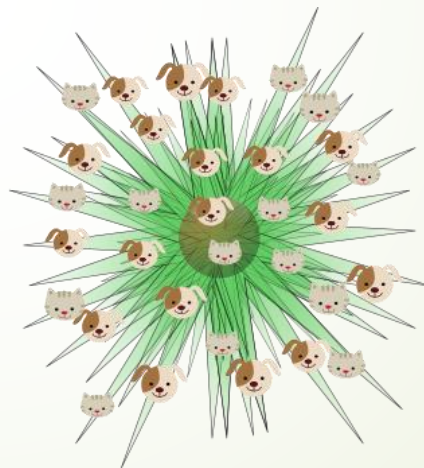
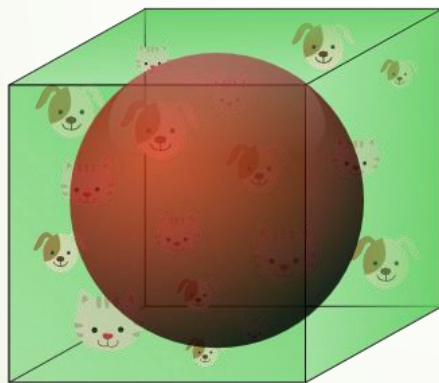
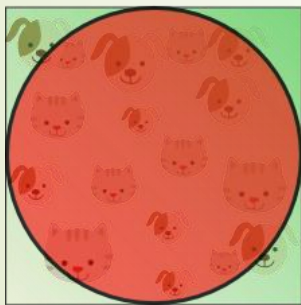
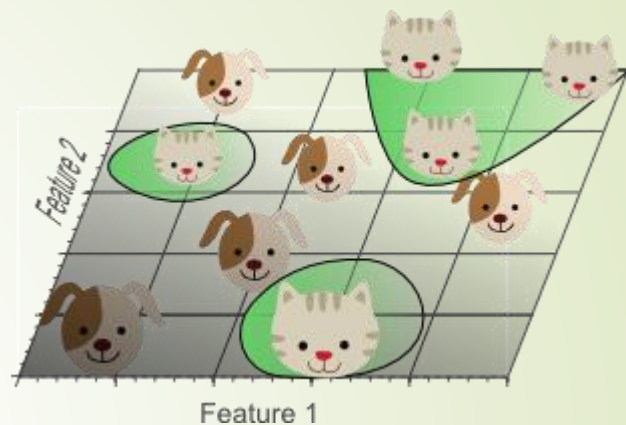
- ➡ 特征的信息量与区分度
- ➡ 特征正交化与PCA降维
- ➡ 数据分析应用与数据可视化

数据特征与数据维度

	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

维度灾难

1. 高维度下，数据样本稀疏，难以做到密采样，易过拟合
2. 在高维空间，特征间的某些距离测量逐渐失效



回顾两个数学概念：概率 - 信息量

➡ 概率： $P_i = F_i / \sum_i F_i$ （古典概型，也称频率模型）

➡ 信息量： $H_i = -\log P_i$

概率越小，信息量越大，概率越大，信息量越小

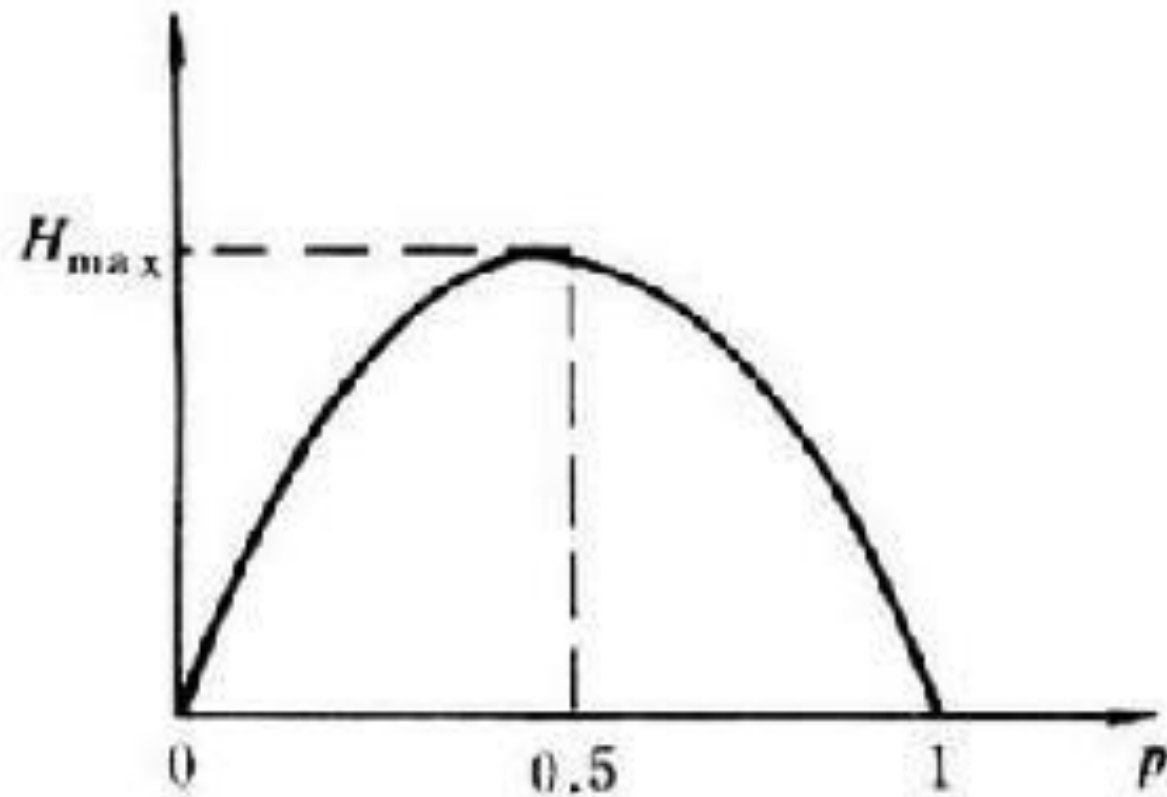
再进一步：编码系统的信息量（信息熵）

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

2元编码系统均匀分布的信息熵：

$$H_2 = 2^* (-1/2 \log (1/2)) = 1 \text{ bit}$$

4元编码系统均匀分布的信息熵
= ?

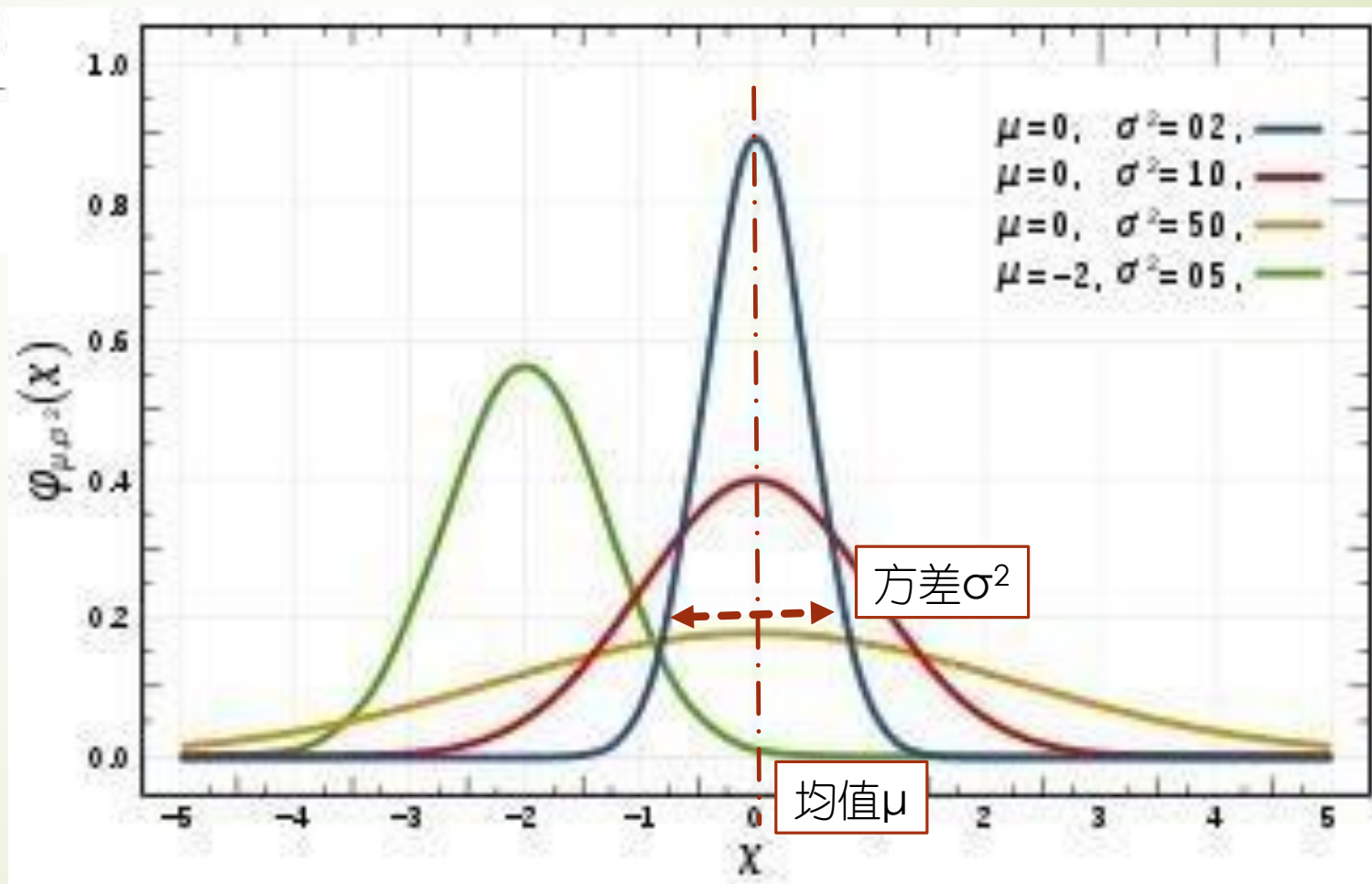


二元信源的熵函数

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



特征的协方差矩阵

- $C_{\vec{X}}(i; j) = Cov_{X_i, X_j} = E \left((X_i - E(X_i)) (X_j - E(X_j)) \right)$
- 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 $X = \begin{pmatrix} -\vec{X}^{(1)} & - \\ -\vec{X}^{(2)} & - \\ \dots & \\ -\vec{X}^{(N)} & - \end{pmatrix}$

- 可以估计协方差矩阵如下:

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

则

$$C = X^T X / N$$

协方差矩阵的物理意义

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

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

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

新特征的方差

新特征空间的基底

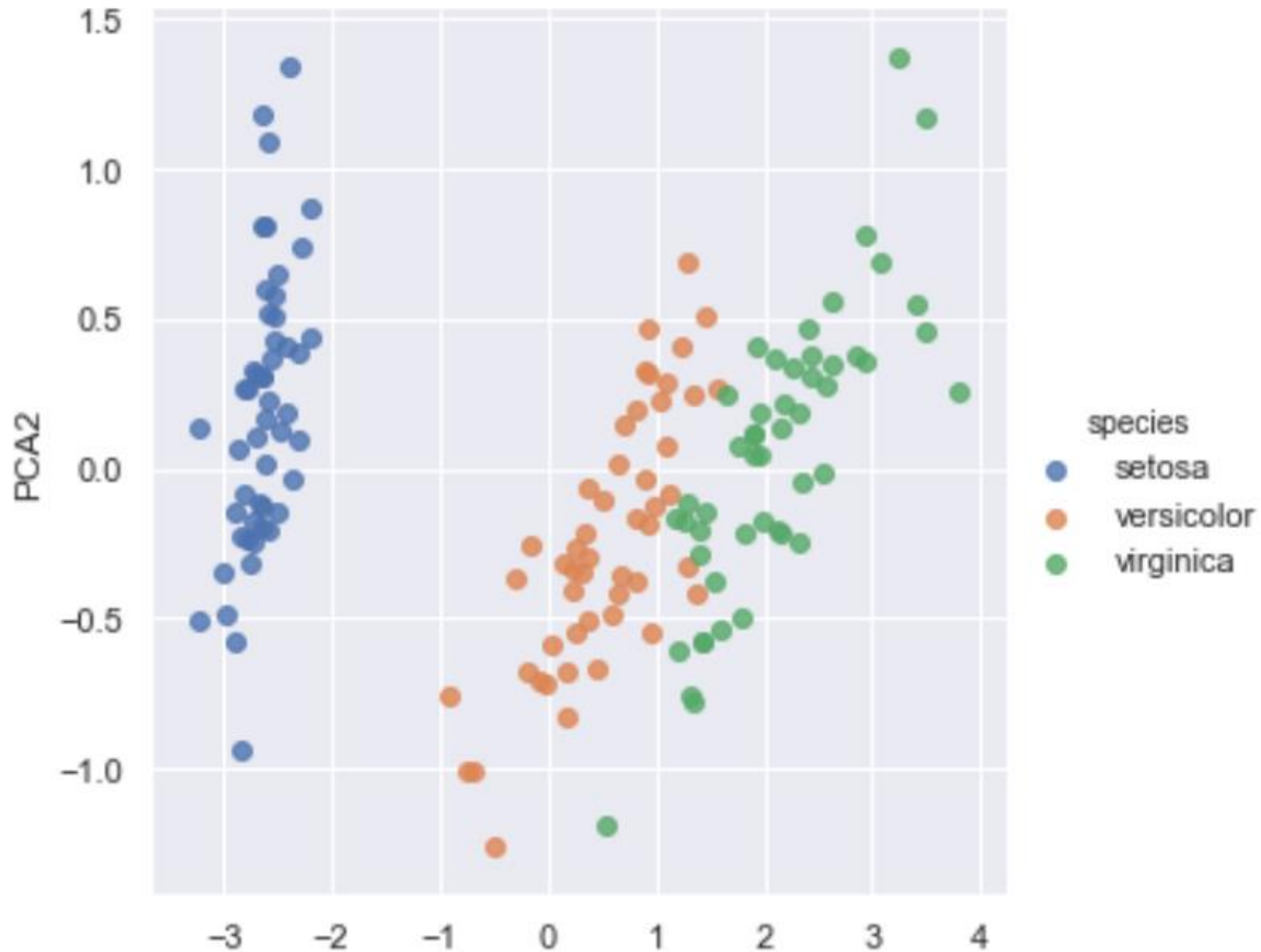
PCA数据特征降维

► In [78]:

```
1 from sklearn.decomposition import PCA # 1. Choose the model class
2 model = PCA(n_components=2)           # 2. Instantiate the model with hyperp
3 model.fit(X_iris)                     # 3. Fit to data. Notice y is not spec
4 X_2D = model.transform(X_iris)        # 4. Transform the data to two dimensi
5 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],

```
1 iris['PCA1'] = X_2D[:, 0]
2 iris['PCA2'] = X_2D[:, 1]
3 sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False)
```



主成分分析与聚类

```
import pandas as pd
```

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

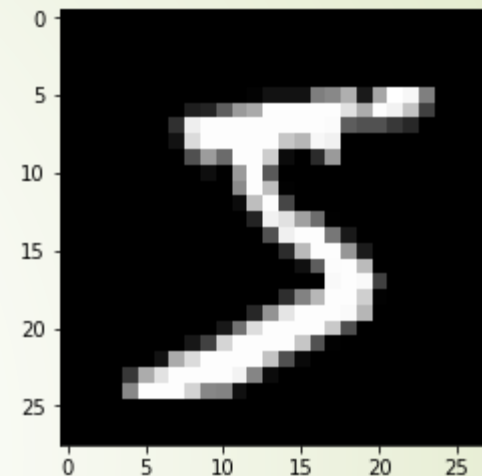
```
(42000, 785)
```

```
: target = train['label']  
train = train.drop("label", axis=1)  
X = train[:6000].values  
Target = target[:6000]  
print(X.shape)
```

```
(6000, 784)
```

← 提取出data, label

	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 — 手写数字识别

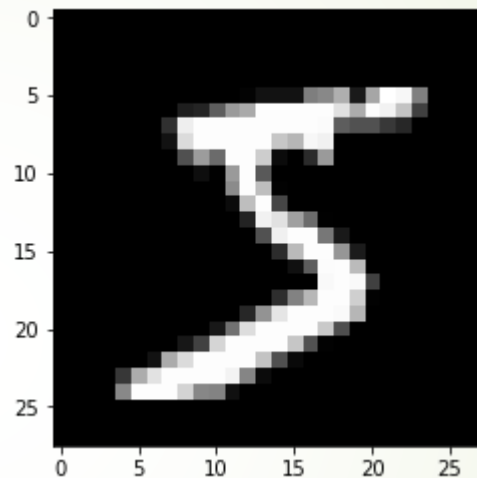
标准化数据，保证每个维度的数据方差是1，均值为0，使得预测结果不会被某些数值大的值主导，保证‘公平’

计算特征值和特征向量

从特征值由大到小排列

计算每个方差所占能量的比例

聚类与识别



5

```
from sklearn.cluster import KMeans # KMeans clustering
```

```
kmeans = KMeans(n_clusters=10)
```

```
X_clustered = kmeans.fit_predict(tsne_results)
```

```
fig = plt.figure()
```

```
ax1 = fig.add_subplot(111)
```

```
#设置标题
```

```
ax1.set_title('KMeans Clustering (LDA)')
```

```
#设置X轴标签
```

```
plt.xlabel('First Principal Component')
```

```
#设置Y轴标签
```

```
plt.ylabel('Second Principal Component')
```

```
#画散点图
```

```
ax1.scatter(tsne_results[:,0], tsne_results[:,1], c = X_clustered, cmap='jet', marker = 'o')
```

```
#显示所画的图
```

```
plt.show()
```

