数据分析建模基础

字符串编码格式回顾：

- ASCII：早起计算机保存英文字符的编码方式
- GB2312：对 ASCII 的中文扩展
- GBK/GB18030：包括了 GB2312 的所有内容，同时又增加了近 20000 个新的汉字和符号
- Unicode：包括了全球的符合和编码。每个字符用 3~4 个字节表示，浪费空间
- UTF-8：可变长的编码方式，在互联网上使用最广泛的一种 Unicode 的实现方式，根据语种决定字符长度，如一个汉字 3 个字节，一个字母 1 个字节，也是 Linux 环境下默认编码格式。

DIKW 模型和数据工程

DIKW 体系

DIKW 体系是关于数据、信息、知识及智慧的体系，可以追溯至托马斯·斯特尔那斯·艾略特所写的诗--《岩石》。在首段，他写道：“我们在哪里丢失了知识中的智慧？又在哪里丢失了信息中的知识？”（Where is the wisdom we have lost in knowledge？ / Where is the knowledge we have lost in information？）。

1982 年 12 月，美国教育家哈蓝·克利夫兰引用艾略特的这些诗句在其出版的《未来主义者》一书提出了“信息即资源”（Information as a Resource）的主张。

其后，教育家米兰·瑟兰尼、管理思想家罗素·艾可夫进一步对此理论发扬光大，前者在 1987 年撰写了《管理支援系统：迈向整合知识管理》（Management Support Systems: Towards Integrated Knowledge Management ），后者在 1989 年撰写了《从数据到智慧》（“From Data to Wisdom”，Human Systems Management）。

数据工程领域中的 DIKW 体系

- D：Data (数据)，是 DIKW 体系中最低级的材料，一般指原始数据，包含（或不包含）有用的信息。
- I：Information (信息)，作为一个概念，信息有着多种多样的含义。在数据工程里，表示由数据工程师（使用相关工具）或者 数据科学家（使用数学方

法），按照某种特定规则，对原始数据进行整合提取后，找出来的更高层数据（具体数据）。

- K：Knowledge (知识)，是对某个主题的确定认识，并且这些认识拥有潜在的能力为特定目的而使用。在数据工程里，表示对信息进行针对性的实用化，让提取的信息可以用于商业应用或学术研究。

- W：Wisdom (智慧)，表示对知识进行独立的思考分析，得出的某些结论。在数据工程里，工程师和科学家做了大量的工作用计算机程序尽可能多地提取了价值（I/K），然而真正要从数据中洞察出更高的价值，甚至能够对未来的情况进行预测，则需要数据分析师。

数据工程 领域职业划分

数据工程是一整套对数据（D）进行采集、处理、提取价值（变为I或K）的过程。

首先介绍一下相关的几种角色： Data Engineer（数据工程师）, Data Scientist（数据科学家）, Data Analyst（数据分析师）。 这三个角色任务重叠性高，要求合作密切，但各负责的领域稍有不同。大部分公司里的这些角色都会根据每个人本身的技能长短而身兼数职， 所以有时候比较难以区分：

- Data Engineer 数据工程师： 分析数据少不了需要运用计算机和各种工具自动化数据处理的过程， 包括数据格式转换， 储存， 更新， 查询。 数据工程师的工作就是开发工具完成自动化的过程， 属于 基础设施/工具（Infrastructure/Tools）层。

  但是这个角色出现的频率不多 ，因为有现成的 MySQL, Oracle 等数据库技术， 很多大公司只需要 DBA 就足够了。而 Hadoop, MongoDB 等 NoSQL 技术的开源， 更是使在大数据的场景下都没有太多 数据工程师 的事，一般都是交给 数据科学家 。

- Data Scientist 数据科学家： 数据科学家是与数学相结合的中间角色， 需要用数学方法处理原始数据找出肉眼看不到的更高层数据， 一般是运用 统计机器学习（Statistical Machine Learning）或者 深度学习（Deep Learning）。

  有人称 Data Scientist 为 编程统计学家（Programming Statistician），因为他们需要有很好的统计学基础，但也需要参与程序的开发（基于 Infrastructure 之上），而现在很多很多的数据科学家 职位都要求身兼数据工程师。 数据科学家 是把 D 转为 I 或 K 的主力军。

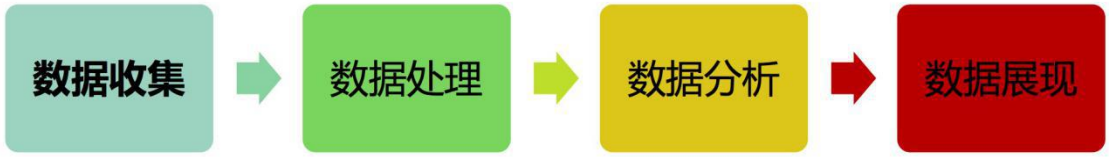
- Data Analyst 数据分析师： 数据工程师和数据科学家做了大量的工作，用计算机程序尽可能多地提取了价值（I/K），然而真正要从数据中洞察出更高的价值， 则需要依靠丰富的行业经验和洞察力， 这些都需要人力的干预。

  Data Analyst 需要的是对所在业务有深刻了解， 能熟练运用手上的工具（无论是 Excel， SPSS 也好， Python/R 也好，工程师给你开发的工具也好，必要时还要能自己充当工程师和科学家，力尽所能得到自己需要的工具），有针对性地对数据作分析，并且需要把发现的成果向其他职能部门呈现出来，最终变为行动，这就是把数据最终得出 Wisdom。

什么是数据分析

百度百科：数据分析是指用适当的统计分析方法对收集来的大量数据进行分析，提取有用信息和形成结论而对数据加以详细研究和概括总结的过程。这一过程也是质量管理体系的支持过程。在实用中，数据分析可帮助人们作出判断，以便采取适当行动。

过程：



1. 数据收集：本地数据或者网络数据的采集与操作.
2. 数据处理：数据的规整，按照某种格式进行整合存储。
3. 数据分析：数据的科学计算，使用相关数据工具进行分析。
4. 数据展现：数据可视化，使用相关工具对分析出的数据进行展示。

数据分析工具

- SAS：SAS（STATISTICAL ANALYSIS SYSTEM，简称 SAS）公司开发的统计分析软件，是一个功能强大的数据库整合平台。价格昂贵，银行或者大企业才买的起，做离线的分析或者模型用。
- SPSS：SPSS（Statistical Product and Service Solutions，统计产品与服务解决方案）是 IBM 公司推出的一系列用于统计学分析运算、数据挖掘、预测分析和决策支持任务的产品，迄今已有 40 余年的成长历史，价格昂贵。
- R/MATLAB：适合做学术性质的数据分析，在实际应用上需要额外转换为 Python 或 Scala 来实现，而且 MATLAB（MathWorks 公司出品的商业数学软件）是收费的。

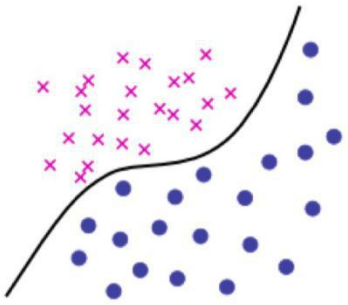
- Scala：是一门函数式编程语言，熟练使用后开发效率较高，配合 Spark 适合大规模的数据分析和处理，Scala 的运行环境是 JVM。
- Python：Python 在数据工程领域和机器学习领域有很多成熟的框架和算法库，完全可以只用 Python 就可以构建以数据为中心的应用程序。在数据工程领域和机器学习领域，Python 非常非常流行。

数据建模基础

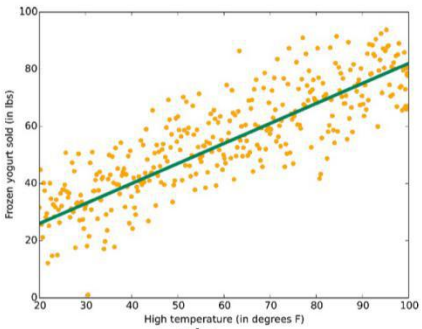
大数据分析场景和模型应用

数据分析建模需要先明确业务需求，然后选择是 描述型分析 还是 预测型分析。

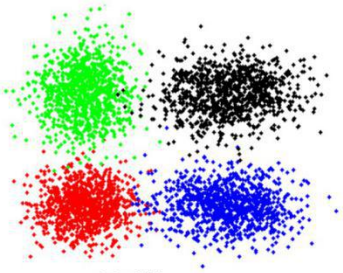
- 如果分析的目的是描述目标行为模式，就采用描述型数据分析，描述型分析就考虑关联规则、序列规则 、聚类等模型。
- 如果是预测型数据分析，就是量化未来一段时间内，某个事件的发生概率。有两大预测分析模型， 分类预测和回归预测。



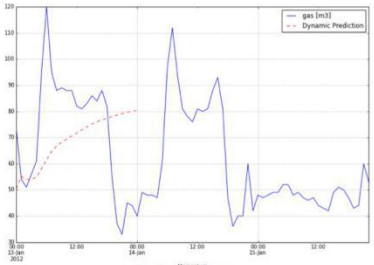
分类



回归



聚类



时序分析

分类与回归

- 分类：是通过已有的训练样本去训练得到一个最优模型，再利用这个模型将输入映射为相应的输出，对输出进行简单的判断从而实现分类的目的，也就具有了对未知数据进行分类的能力。
- 回归：是基于观测数据建立变量间适当的依赖关系，以分析数据内在的规律，得到响应的判断。并可用于预报、控制等问题。

应用

信用卡申请人风险评估、预测公司业务增长量、预测房价，未来的天气情况等

原理

- 回归：用属性的 历史数据 预测未来趋势。算法首先假设一些已知类型的函数可以匹配目标数据，然后分析匹配后的误差，确定一个与目标数据匹配程度最好的函数。回归是对真实值的一种 逼近预测。
- 分类：将数据映射到 预先定义的 群组或类。算法要求基于数据 特征值 来定义类别，把具有某些特征的数据项映射到给定的某个类别上。分类并没有逼近的概念，最终正确结果只有一个。 在机器学习方法里，分类属于监督学习。

区别

分类模型采用 离散预测值，回归模型采用 连续的预测值。

聚类

- 聚类：就是将相似的事物聚集在一起，不相似的事物划分到不同的类别的过程。
- 聚类分析：又称群分析，它是研究（样品或指标）分类问题的一种统计分析方法，同时也是数据挖掘的一个重要算法。

应用

根据症状归纳特定疾病、发现信用卡高级用户、根据上网行为对客户分群从而进行精确营销等。

原理

在没有给定划分类的情况下，根据信息相似度进行信息聚类。

聚类的输入是一组 未被标记的数据，根据样本特征的距离或相似度进行划分。划分原则是保持最大的组内相似性和最小的组间相似性。

不同于分类，聚类事先 没有任何训练样本，直接对数据进行建模。聚类分析的目标，就是在相似的基础上收集数据来分类。 在机器学习方法里，聚类属于无监督学习。

时序模型

不管在哪个领域中（如金融学、经济学、生态学、神经科学、物理学等），时间序列（time series）数据都是一种重要的结构化数据形式。在多个时间点观察或测量到的任何事物，都可以形成一段时间序列。时间序列大多都是固定频率的，数据点将根据某种规律定期出现。

应用

下个季度的商品销量或库存量是多少？明天用电量是多少？今天的北京地铁 13 号线的人流情况？

原理

描述 基于时间或其他序列的 经常发生的规律或趋势，并对其建模。 与回归一样，用已知的数据预测未来的值，但这些数据的区别是 变量所处时间的不同。重点考察数据之间在 时间维度上的关联性。

常见的数据分析应用场景

市场营销

- 营销响应分析建模(逻辑回归，决策树)
- 净提升度分析建模(关联规则)
- 客户保有分析建模(卡普兰梅尔分析，神经网络)
- 购物蓝分析(关联分析 Apriori)
- 自动推荐系统(协同过滤推荐，基于内容推荐，基于人口统计推荐，基于知识推荐，组合推荐，关联规则)
- 客户细分(聚类)
- 流失预测(逻辑回归)

风险管理

- 客户信用风险评分(SVM，决策树，神经网络)
- 市场风险评分建模(逻辑回归和决策树)
- 运营风险评分建模(SVM)

- 欺诈检测(决策树，聚类，社交网络)

科学计算工具 NumPy

Numpy（Numerical Python）

Numpy：提供了一个在 Python 中做科学计算的基础库，重在数值计算，主要用于多维数组（矩阵）处理的库。用来存储和处理大型矩阵，比 Python 自身的嵌套列表结构要高效的多。本身是由 C 语言开发，是个很基础的扩展，Python 其余的科学计算扩展大部分都是以此为基础。

- 高性能科学计算和数据分析的基础包
- ndarray，多维数组（矩阵），具有矢量运算能力，快速、节省空间
- 矩阵运算，无需循环，可完成类似 Matlab 中的矢量运算
- 线性代数、随机数生成
- `import numpy as np`

Scipy

Scipy ：基于 Numpy 提供了一个在 Python 中做科学计算的工具集，专为科学和工程设计的 Python 工具包。主要应用于统计、优化、整合、线性代数模块、傅里叶变换、信号和图像处理、常微分方程求解、稀疏矩阵等，在数学系或者工程系相对用的多一些，和数据处理的关系不大， 我们知道即可，这里不做讲解。

- 在 NumPy 库的基础上增加了众多的数学、科学及工程常用的库函数
- 线性代数、常微分方程求解、信号处理、图像处理
- 一般的数据处理 numpy 已经够用
- `import scipy as sp`

参考学习资料：

Python、NumPy 和 SciPy 介绍：http://cs231n.github.io/python-numpy-tutorial

NumPy 和 SciPy 快速入门：https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

ndarray 多维数组(N Dimension Array)

NumPy 数组是一个多维的数组对象（矩阵），称为 ndarray，具有矢量算术运算能力和复杂的广播能力，并具有执行速度快和节省空间的特点。

注意：ndarray 的下标从 0 开始，且数组里的所有元素必须是相同类型

ndarray 拥有的属性

- ndim 属性：维度个数
- shape 属性：维度大小
- dtype 属性：数据类型

ndarray 的随机创建

通过随机抽样（numpy.random）生成随机数据。

示例代码：

```
# 导入 numpy，别名 np
import numpy as np

# 生成指定维度大小（3 行 4 列）的随机多维浮点型数据（二维），rand 固定区间 0.0 ~ 1.0
arr = np.random.rand(3, 4)
print(arr)
print(type(arr))

# 生成指定维度大小（3 行 4 列）的随机多维整型数据（二维），randint()可以指定区间（-1, 5）
arr = np.random.randint(-1, 5, size = (3, 4)) # 'size='可省略
print(arr)
print(type(arr))

# 生成指定维度大小（3 行 4 列）的随机多维浮点型数据（二维），uniform()可以指定区间（-1, 5）
arr = np.random.uniform(-1, 5, size = (3, 4)) # 'size='可省略
print(arr)
print(type(arr))

print('维度个数: ', arr.ndim)
print('维度大小: ', arr.shape)
print('数据类型: ', arr.dtype)
```

运行结果：

```
[[ 0.09371338  0.06273976  0.22748452  0.49557778]
 [ 0.30840042  0.35659161  0.54995724  0.018144  ]
 [ 0.94551493  0.70916088  0.58877255  0.90435672]]
<class 'numpy.ndarray'>

[[ 1  3  0  1]
 [ 1  4  4  3]
 [ 2  0 -1 -1]]
<class 'numpy.ndarray'>

[[ 2.25275308  1.67484038 -0.03161878 -0.44635706]
 [ 1.35459097  1.66294159  2.47419548 -0.51144655]
 [ 1.43987571  4.71505054  4.33634358  2.48202309]]
<class 'numpy.ndarray'>
```

维度个数: 2 维度大小: (3, 4) 数据类型: float64

ndarray 的序列创建

np.array(collection)

collection 为 序列型对象(list)、嵌套序列对象(list of list)。

示例代码：

```
# list 序列转换为 ndarray
lis = range(10)
arr = np.array(lis)

print(arr)            # ndarray 数据
print(arr.ndim)        # 维度个数
print(arr.shape)    # 维度大小

# list of list 嵌套序列转换为 ndarray
lis_lis = [range(10), range(10)]
arr = np.array(lis_lis)

print(arr)            # ndarray 数据
print(arr.ndim)        # 维度个数
print(arr.shape)    # 维度大小
```

运行结果：

```
# list 序列转换为 ndarray
[0 1 2 3 4 5 6 7 8 9]
1
(10,)
```

```
# list of list 嵌套序列转换为 ndarray
[[0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]]
2
(2, 10)
```

np.zeros()

指定大小的全 0 数组。注意：第一个参数是元组，用来指定大小，如(3, 4)。

np.ones()

指定大小的全 1 数组。注意：第一个参数是元组，用来指定大小，如(3, 4)。

np.empty()

初始化数组，不是总是返回全 0，有时返回的是未初始的随机值（内存里的随机值）。

示例代码（2、3、4）：

```
# np.zeros
zeros_arr = np.zeros((3, 4))

# np.ones
ones_arr = np.ones((2, 3))

# np.empty
empty_arr = np.empty((3, 3))

# np.empty 指定数据类型
empty_int_arr = np.empty((3, 3), int)

print('------zeros_arr-------')
print(zeros_arr)

print('\n------ones_arr-------')
print(ones_arr)

print('\n------empty_arr-------')
print(empty_arr)

print('\n------empty_int_arr-------')
print(empty_int_arr)
```

运行结果：

```
------zeros_arr-------
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]

------ones_arr-------
[[ 1.  1.  1.]
 [ 1.  1.  1.]]

------empty_arr-------
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]

------empty_int_arr-------
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

np.arange() 和 reshape()

arange() 类似 python 的 range() ，创建一个一维 ndarray 数组。

reshape() 将 重新调整数组的维数。

示例代码（5）：

```
# np.arange()
arr = np.arange(15) # 15 个元素的 一维数组
print(arr)
print(arr.reshape(3, 5)) # 3x5 个元素的 二维数组
print(arr.reshape(1, 3, 5)) # 1x3x5 个元素的 三维数组
```

运行结果：

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]

[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]

[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]
  [10 11 12 13 14]]]
```

np.arange() 和 random.shuffle()

random.shuffle() 将打乱数组序列（类似于洗牌）。

示例代码（6）：

```
arr = np.arange(15)
print(arr)

np.random.shuffle(arr)
print(arr)
print(arr.reshape(3,5))
```

运行结果：

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]

[ 5  8  1  7  4  0 12  9 11  2 13 14 10  3  6]

[[ 5  8  1  7  4]
 [ 0 12  9 11  2]
 [13 14 10  3  6]]
```

ndarray 的数据类型

dtype 参数

指定数组的数据类型，类型名+位数，如 float64, int32

astype 方法

转换数组的数据类型

示例代码（1、2）：

```
# 初始化 3 行 4 列数组，数据类型为 float64
zeros_float_arr = np.zeros((3, 4), dtype=np.float64)
print(zeros_float_arr)
print(zeros_float_arr.dtype)

# astype 转换数据类型，将已有的数组的数据类型转换为 int32
zeros_int_arr = zeros_float_arr.astype(np.int32)
print(zeros_int_arr)
print(zeros_int_arr.dtype)
```

运行结果：

```
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
float64

[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
int32
```

ndarray 进行数据处理

ndarray 的矩阵运算

数组是编程中的概念，矩阵、矢量是数学概念。

在计算机编程中，矩阵可以用数组形式定义，矢量可以用结构定义!

矢量运算：

相同大小的数组间运算应用在元素上

示例代码（1）：

```
# 矢量与矢量运算

arr = np.array([[1, 2, 3],

                [4, 5, 6]])



print("元素相乘：")

print(arr * arr)



print("矩阵相加：")

print(arr + arr)
```

运行结果：

元素相乘：

```
[[ 1  4  9]

 [16 25 36]]
```

矩阵相加：

```
[[ 2  4  6]

 [ 8 10 12]]
```

矢量和标量运算：

"广播" - 将标量"广播"到各个元素

示例代码（2）：

```
# 矢量与标量运算
print(1. / arr)
print(2. * arr)
```

运行结果：

```
[[ 1.          0.5         0.33333333]
 [ 0.25        0.2         0.16666667]]

[[  2.   4.   6.]
 [  8.  10.  12.]]
```

ndarray 的索引与切片

一维数组的索引与切片

与 Python 的列表索引功能相似

示例代码（1）：

```
# 一维数组
arr1 = np.arange(10)
print(arr1)
print(arr1[2:5])
```

运行结果：

```
[0 1 2 3 4 5 6 7 8 9]
[2 3 4]
#### 多维数组的索引与切片：
```

arr[r1:r2, c1:c2]

arr[1,1] 等价 arr[1][1]

[:] 代表某个维度的数据

示例代码（2）：

```
# 多维数组
arr2 = np.arange(12).reshape(3,4)
print(arr2)

print(arr2[1])

print(arr2[0:2, 2:])

print(arr2[:, 1:3])
```

运行结果：

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[4 5 6 7]

[[2 3]
 [6 7]]

[[ 1  2]
 [ 5  6]
 [ 9 10]]
```

条件索引

布尔值多维数组：arr[condition]，condition 也可以是多个条件组合。

注意，多个条件组合要使用 & | 连接，而不是 Python 的 and or。

示例代码（3）：

```
# 条件索引

# 找出 data_arr 中 2005 年后的数据
data_arr = np.random.rand(3,3)
print(data_arr)
```

```
year_arr = np.array([[2000, 2001, 2000],
                     [2005, 2002, 2009],
                     [2001, 2003, 2010]])

is_year_after_2005 = year_arr >= 2005
print(is_year_after_2005, is_year_after_2005.dtype)

filtered_arr = data_arr[is_year_after_2005]
print(filtered_arr)

#filtered_arr = data_arr[year_arr >= 2005]
#print(filtered_arr)

# 多个条件
filtered_arr = data_arr[(year_arr <= 2005) & (year_arr % 2 == 0)]
print(filtered_arr)
```

运行结果：

```
[[ 0.53514038  0.93893429  0.1087513 ]
 [ 0.32076215  0.39820313  0.89765765]
 [ 0.6572177   0.71284822  0.15108756]]

[[False False False]
 [ True False  True]
 [False False  True]] bool

[ 0.32076215  0.89765765  0.15108756]

#[ 0.32076215  0.89765765  0.15108756]

[ 0.53514038  0.1087513   0.39820313]
```

ndarray 的维数转换

二维数组直接使用转换函数：transpose()

高维数组转换要指定维度编号参数 (0, 1, 2, …)，注意参数是元组

```
arr = np.random.rand(2,3)     # 2x3 数组
print(arr)
print(arr.transpose()) # 转换为 3x2 数组
```

```
arr3d = np.random.rand(2,3,4) # 2x3x4 数组，2 对应 0，3 对应 1，4 对应 3
print(arr3d)
print(arr3d.transpose((1,0,2))) # 根据维度编号，转为为 3x2x4 数组
```

运行结果：

```
# 二维数组转换

# 转换前：

[[ 0.50020075  0.88897914  0.18656499]

 [ 0.32765696  0.94564495  0.16549632]]


# 转换后：

[[ 0.50020075  0.32765696]

 [ 0.88897914  0.94564495]

 [ 0.18656499  0.16549632]]



# 高维数组转换

# 转换前：

[[[ 0.91281153  0.61213743  0.16214062  0.73380458]

  [ 0.45539155  0.04232412  0.82857746  0.35097793]

  [ 0.70418988  0.78075814  0.70963972  0.63774692]]


 [[ 0.17772347  0.64875514  0.48422954  0.86919646]

  [ 0.92771033  0.51518773  0.82679073  0.18469917]
```

```
  [ 0.37260457  0.49041953  0.96221477  0.16300198]]]


# 转换后：

[[[ 0.91281153  0.61213743  0.16214062  0.73380458]
  [ 0.17772347  0.64875514  0.48422954  0.86919646]]

 [[ 0.45539155  0.04232412  0.82857746  0.35097793]
  [ 0.92771033  0.51518773  0.82679073  0.18469917]]

 [[ 0.70418988  0.78075814  0.70963972  0.63774692]
  [ 0.37260457  0.49041953  0.96221477  0.16300198]]]
```

元素计算函数

- ceil(): 向上最接近的整数，参数是 number 或 array
- floor(): 向下最接近的整数，参数是 number 或 array
- rint(): 四舍五入，参数是 number 或 array
- isnan(): 判断元素是否为 NaN(Not a Number)，参数是 number 或 array
- multiply(): 元素相乘，参数是 number 或 array
- divide(): 元素相除，参数是 number 或 array
- abs()：元素的绝对值，参数是 number 或 array
- where(condition, x, y): 三元运算符，x if condition else y

示例代码

```
# randn() 返回具有标准正态分布的序列。
arr = np.random.randn(2,3)

print(arr)
```

```
print(np.ceil(arr))

print(np.floor(arr))

print(np.rint(arr))

print(np.isnan(arr))

print(np.multiply(arr, arr))

print(np.divide(arr, arr))

print(np.where(arr > 0, 1, -1))
```

运行结果

```
# print(arr)
[[-0.75803752  0.0314314   1.15323032]
 [ 1.17567832  0.43641395  0.26288021]]

# print(np.ceil(arr))
[[-0.  1.  2.]
 [ 2.  1.  1.]]

# print(np.floor(arr))
[[-1.  0.  1.]
 [ 1.  0.  0.]]

# print(np.rint(arr))
[[-1.  0.  1.]
 [ 1.  0.  0.]]

# print(np.isnan(arr))
[[False False False]
 [False False False]]

# print(np.multiply(arr, arr))
[[  5.16284053e+00   1.77170104e+00   3.04027254e-02]
 [  5.11465231e-03   3.46109263e+00   1.37512421e-02]]

# print(np.divide(arr, arr))
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
```

```
# print(np.where(arr > 0, 1, -1))
[[ 1  1 -1]
 [-1  1  1]]
```

元素统计函数

- np.mean(), np.sum()：所有元素的平均值，所有元素的和，参数是 number 或 array
- np.max(), np.min()：所有元素的最大值，所有元素的最小值，参数是 number 或 array
- np.std(), np.var()：所有元素的标准差，所有元素的方差，参数是 number 或 array
- np.argmax(), np.argmin()：最大值的下标索引值，最小值的下标索引值，参数是 number 或 array
- np.cumsum(), np.cumprod()：返回一个一维数组，每个元素都是之前所有元素的 累加和 和 累乘积，参数是 number 或 array
- 多维数组默认统计全部维度，axis 参数可以按指定轴心统计，值为 0 则按列统计，值为 1 则按行统计。 示例代码：

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

print(np.cumsum(arr)) # 返回一个一维数组，每个元素都是之前所有元素的 累加和

print(np.sum(arr)) # 所有元素的和

print(np.sum(arr, axis=0)) # 数组的按列统计和

print(np.sum(arr, axis=1)) # 数组的按行统计和
```

运行结果：

```
# print(arr)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

# print(np.cumsum(arr))
[ 0  1  3  6 10 15 21 28 36 45 55 66]

# print(np.sum(arr)) # 所有元素的和
66
```

```
# print(np.sum(arr, axis=0)) # 0 表示对数组的每一列的统计和
[12 15 18 21]

# print(np.sum(arr, axis=1)) # 1 表示数组的每一行的统计和
[ 6 22 38]
```

元素判断函数

- np.any(): 至少有一个元素满足指定条件，返回 True
- np.all(): 所有的元素满足指定条件，返回 True 示例代码：

```
arr = np.random.randn(2,3)
print(arr)

print(np.any(arr > 0))
print(np.all(arr > 0))
```

运行结果：

```
[[ 0.05075769 -1.31919688 -1.80636984]
 [-1.29317016 -1.3336612  -0.19316432]]

True
False
```

元素去重排序函数

np.unique():找到唯一值并返回排序结果，类似于 Python 的 set 集合

示例代码：

```
arr = np.array([[1, 2, 1], [2, 3, 4]])
print(arr)

print(np.unique(arr))
```

运行结果：

```
[[1 2 1]

 [2 3 4]]



[1 2 3 4]
```

案例:2016 年美国总统大选民意调查数据统计

```
import numpy as np

# 读取列名，即第一行数据

with open(filename, 'r') as f:

    col_names_str = f.readline()[:-1] # [:-1]表示不读取末尾的换行符'\n'



# 将字符串拆分，并组成列表

col_name_lst = col_names_str.split(',')



# 使用的列名：结束时间，克林顿原始票数，川普原始票数，克林顿调整后票数，川普调整后票数

use_col_name_lst = ['enddate', 'rawpoll_clinton', 'rawpoll_trump','adjpoll_clinton', 'adjpoll_trump']



# 获取相应列名的索引号

use_col_index_lst = [col_name_lst.index(use_col_name) for use_col_name in use_col_name_lst]



# 通过 genfromtxt()读取本地 csv 文件，

data_array = np.genfromtxt(filename,       # 文件名

                           delimiter=',', # 分隔符

                           #skiprows=1,     # 跳过第一行，即跳过列名

                           dtype=str,      # 数据类型，数据不再是 Unicode 字符串

                           usecols=use_col_index_lst)# 指定读取的列索引号
```

```
# genfromtxt() 不能通过 skiprows 跳过第一行的

# ['enddate' 'rawpoll_clinton' 'rawpoll_trump' 'adjpoll_clinton' 'adjpoll_trump']


# 去掉第一行

data_array = data_array[1:]


# 打印 ndarray 数据

print(data_array[1:], data_array.shape)
```