《人工智能与Python程序设计》—— 科学计算

人工智能与Python程序设计 教研组

Python和Numpy进阶知识补充

提纲

- 变量、对象引用

变量引用对象

- 变量赋值：本质上是将变量“贴”在对象上，运行使用变量来访问、修改对象
  - 对象是一块内存空间，内存空间里存储它们所表示的值；
  - 变量是到内存空间的一个标签或引用，也就是拥有指向对象存储的空间；
  - 引用就是自动形成的从变量到对象的映射关系（类似指针）
    - 引用可以看成对象的别名，通过别名可以直接操纵对象
    - 但与C语言中的指针不同，我们无法直接修改指针的值
      - 只能通过变量访问对象，或通过赋值更改变量指向的对象
  - **赋值是将一个变量标签与一个实际对象建立关联的过程**
  - **python中的变量没有类型信息，类型的概念存在于对象中而不是变量中**
    - 思考：Python中**类型**到底指什么?

可变对象和不可变对象

- 可变对象可以被修改，包括列表list、字典dict、集合set
- 不可变对象无法修改，包括数字、字符串str，元组tuple
- 可变对象与不可变对象：
  - 相等性（==，is）
  - 赋值（=）和增强赋值（如:+=)语句实际的语义是什么?
  - 组合数据类型内保存的也是对对象的引用
    - a[x] = y的实际语义是什么?
  - 不可变数据类型（以元组tuple为例）
    - 如果引用的元素是可变的，即便元组本身不可变，元素依然可变。
    - 也就是说，元组的不可变性其实是指 tuple 数据结构的物理内容（即保存的引用）不可变，与引用的对象无关。

函数的参数作为引用

- Python 唯一支持的参数传递模式是传递对象的引用的复本
  - 共享传参（call by sharing）

- 共享传参指函数的各个形式参数获得实参中各个引用的副本。也就是说，函数内部的形参是实参的别名。
  - 传了一个指向相同对象的“标签”

- 函数可能会修改接收到的任何可变对象

```
def f(a, b):
    a += b
    return a

x = 1
y = 2
print(f(x, y))
print(x, y)

a = [1, 2]
b = [3, 4]
print(f(a, b))
print(a, b)

t = (10, 20)
u = (30, 40)
print(f(t, u))
print(t, u)
```

```
3
1 2
[1, 2, 3, 4]
[1, 2, 3, 4] [3, 4]
(10, 20, 30, 40)
(10, 20) (30, 40)
```

函数的参数作为引用

- **思考**：什么时候会修改函数外的数值?

```
>>> a = 523432
>>> id(a)
37656816
>>> def f(x):
...     x *= 3
...     return x
...
>>> f(a)
1570296
>>> id(a)
37656816
>>> print(a)
523432
```

```
>>> b = [1]
>>> id(b)
39823360
>>> f(b)
[1, 1, 1]
>>> id(b)
39823360
>>> print(b)
[1, 1, 1]
>>>
```

当传入参数指向可变对象时才可能会被修改

危险的可变默认值

- 幽灵车

```
class HauntedBus:
    '''
     备受折磨的幽灵车
    '''
    def __init__(self, passengers=[]):
        self.passengers = passengers

    def pick(self, name):
        self.passengers.append(name)

    def drop(self, name):
        self.passengers.remove(name)

bus1 = HauntedBus(['Alice', 'Bill'])
print('bus1上的乘客：', bus1.passengers)
bus1.pick('Charlie')    #bus1上来一名乘客Charile
bus1.drop('Alice')     #bus1下去一名乘客Alice
print('bus1上的乘客：', bus1.passengers)    #打印bus1上的乘客

bus2 = HauntedBus()     #实例化bus2
bus2.pick('Carrie')     #bus2上来一名课程Carrie
print('bus2上的乘客：', bus2.passengers)
```

```
bus3 = HauntedBus()
print('bus3上的乘客：', bus3.passengers)
bus3.pick('Dave')
print('bus2上的乘客：', bus2.passengers)
#登录到bus3上的乘客Dava跑到了bus2上面

print('bus2是否为bus3的对象：', bus2.passengers is bus3.passengers)
print('bus1上的乘客：', bus1.passengers)
```

```
bus1上的乘客： ['Alice', 'Bill']
bus1上的乘客： ['Bill', 'Charlie']
bus2上的乘客： ['Carrie']
bus3上的乘客： ['Carrie']
bus2上的乘客： ['Carrie', 'Dave']
bus2是否为bus3的对象： True
bus1上的乘客： ['Bill', 'Charlie']
```

幽灵车

- 实例化 HauntedBus 时，如果传入乘客，会按预期运作。
- 如果不为 HauntedBus 指定乘客的话，奇怪的事就发生了，这是因为 self.passengers 变成了 passengers 参数默认值的别名。
- 出现这个问题的根源是，默认值在定义函数时计算（通常在加载模块时），因此默认值变成了函数对象的属性。
- 如果默认值是可变对象，而且修改了它的值，那么后续的函数调用都会受到影响。

Python AI
Numpy与科学计算

提纲

- 第三方库安装
- numpy库
- 线性回归

计算生态

- Python 语言有9万多个第三方库，形成了庞大的编程计算生态：
  - Python 语言从诞生之初致力于开源开放
  - Python 官方网站提供了第三方库索引功能
  - https://pypi.python.org/pypi
  - 产业界广泛利用可重用资源快速构建应用已经是主流产品开发方式

- 很多用C、C++等语言编写的专业库可以经过简单的接口封装供Python 语言程序调用

第三方库

- Python第三方程序包括库（library）、模块（ module）、类（class）和程序包（Package）等多种命名
- 统称为“库”
- Python 内置的库称为标准库，其他库称为第三方库
- 编写程序尽可能利用第三方库进行代码复用
- Python 语言的函数库并非都采用Python 语言编写
- 模块编程：利用开源代码和第三方库作为程序的部分或全部模块，搭积木一样编写程序

第三方库安装

- Python第三方库需要安装后才能使用

- 按照安装方式灵活性和难易程度有三个方法：
  - pip 工具安装
  - 自定义安装
  - 文件安装

pip工具安装

- 最常用且最高效的安装方式
  - pip是Python官方提供并维护的在线第三方库安装工具
    - 对于同时安装Python 2 和Python 3 环境的系统，建议采用pip3 命令专门为Python 3 版本安装第三方库

- pip 是Python 内置命令，需要通过命令行执行
- pip -h： 列出pip 常用的子命令

pip使用

- pip 支持安装（install）、下载（download）、卸载（uninstall）、列表（list）、查看（freeze）、查找（search）等一系列安装和维护子命令

- 命令格式：

pip install <拟安装库名>

- 例如，安装pygame 库，pip 工具默认从网络上下载pygame 库安装文件并自动安装到系统中

pip使用

- 使用-U 标签可以更新已安装库的版本，例如，用pip 更新本身：

  pip install -U pip

- 卸载一个库的命令格式如下：

  pip uninstall <拟卸载库名>

- 可以通过list 子命令列出当前系统中已经安装的第三方库，例如：

  pip list

pip使用

- pip 的download 子命令可以下载第三方库的安装包，但并不安装：

  pip download <拟下载库名>

- pip 的search 子命令可以链接搜索库名或摘要中关键字：

  pip search <拟查询关键字>

其它安装方法

- 由于一些历史、技术和政策等原因，有一些第三方库暂时无法用pip安装，需要其他的安装方法

- pip 工具与操作系统也有关系，在Mac OS X 和Linux 等操作系统中，pip 工具几乎可以安装任何Python 第三方库，在Windows 操作系统中，有一些第三方库需要用其他方式安装

自定义安装

- 自定义安装：按照第三方库提供的步骤和方式安装。第三方库都有主页用于维护库的代码和文档。例如：numpy 库的官方主页是：http://www.numpy.org/

- 浏览该网页找到下载链接，如下：http://www.scipy.org/scipylib/download.html

- 根据指示步骤安装。自定义安装一般适合在pip中尚无登记或安装失败的。

文件安装

- 某些第三方库仅提供源代码，通过pip下载文件后无法在Windows系统编译安装，导致安装失败。在Windows 平台下遇到的无法安装第三方库的问题大多属于这类。

- 为了解决这类问题，美国加州大学尔湾分校提供了一个页面，帮助用户获得Windows 可直接安装的第三方库文件，链接地址如下：http://www.lfd.uci.edu/~gohlke/pythonlibs/

- 该地址列出了一批在pip 安装中可能出现问题的第三方库。

文件安装

- 以scipy 为例，首先在页面中找到scipy 库对应的内容
- 选择其中的.whl 文件下载，例如适用于Python 3.8 版本解释器和32位系统的对应文件：scipy-1.6.2-cp38-cp38-win32.whl
- 用pip 命令安装该文件

pip install scipy-1.6.2-cp38-cp38-win32.whl

- .whl 是Python 库的一种打包格式，用于通过pip进行安装，相当于Python 库的安装包文件。
- .whl 文件本质上是一个压缩格式文件，可以通过变化扩展名为.zip查看其中内容。

Python AI
Numpy与科学计算

提纲

- 第三方库安装
- **numpy库**
- 线性回归

科学计算：矩阵

- 科学计算是解决科学和工程中的数学问题利用计算机进行的数值计算
- 组织数据是运算的基础，也是将客观世界数字化的必要手段
- 科学计算以矩阵而不是单一数值为基础，增加了计算密度，能够表达更为复杂的数据运算逻辑

- 数学的矩阵（Matrix）是一个按照长方阵列排列的复数或实数集合
- 矩阵有维度概念，一维矩阵是线性的，类似于列表，二维矩阵是表格状的二维数组

numpy库概述

- Python 标准库中提供了一个array 类型，用于保存数组类型数据，但这个类型不支持多维数据，处理函数也不够丰富，不适合用于做数值运算。
- numpy 是用于处理含有同种元素的多维数组运算的第三方库。
- numpy 已经成为了Python科学计算事实上的标准库。

numpy库概述

- numpy相比于list的优势：
  - 内存效率高
  - 使用更方便
  - 内置了很多强大的功能：FFT、卷积、快速搜索、基本统计、线性代数、直方图等
  - 速度更快

```
%timeit sum(range(10000000))
```

```
342 ms ± 14.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
from numpy import arange
%timeit arange(10000000).sum()
```

```
16.2 ms ± 149 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

numpy库概述

- numpy 库处理的最基础数据类型是由同种元素构成的多维数组（ndarray），简称“数组”。
- 数组中所有元素的**类型**必须相同。
- 数组中元素可以用整数索引，序号从0开始。
- ndarray 类型的维度(dimensions)叫做轴(axes)。
- 由于numpy库中函数较多且命名容易与常用命名混淆，建议采用如下方式引用numpy 库：

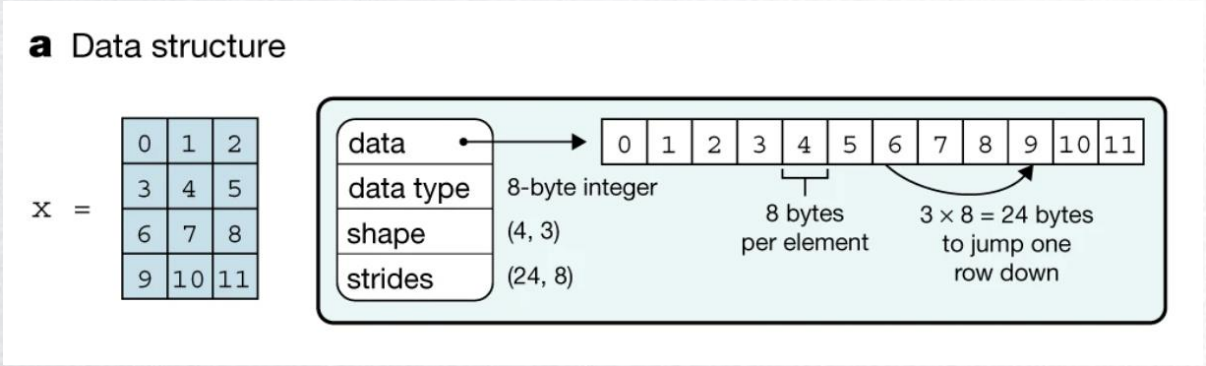
import numpy as np

  - 其中，as 保留字与import 一起使用能够改变后续代码中库的命名空间，有助于提高代码可读性。
  - 在程序的后续部分中，np 代替numpy。

Numpy数组

- Numpy数组（ndarray）是Numpy库中最核心的数据类型
  - 支持对多维数组（又叫张量（Tensor））的高效存储和访问
  - 其结构如下：

```
In [3]: import numpy as np
        x = np.arange(12).reshape((4,3))
        print(type(x))
        print(x)

        <class 'numpy.ndarray'>
        [[ 0  1  2]
         [ 3  4  5]
         [ 6  7  8]
         [ 9 10 11]]
```



(from Array programming with NumPy, Nature volume 585, pages357–362(2020))

  - 与python内置list不同：
    - 所有元素都是一个类型的，由data type(x.dtype)决定
    - 数据保存在一段连续的内存空间中（与C语言中数组类似）
    - 目的：节省存储空间，提升访问效率

numpy库数组创建

函数	描述
np.array([x,y,z], dtype=int)	从 Python 列表和元组创造数组
np.arange(x,y,i)	创建一个由 x 到 y，以 i 为步长的数组
np.linspace(x,y,n)	创建一个由 x 到 y，等分成 n 个元素的数组
np.indices((m,n))	创建一个 m 行 n 列的矩阵
np.random.rand(m,n)	创建一个 m 行 n 列的随机数组
np.ones((m,n),dtype)	创建一个 m 行 n 列全 1 的数组，dtype 是数据类型
np.empty((m,n),dtype)	创建一个 m 行 n 列全 0 的数组，dtype 是数据类型

numpy库数组创建

- 数组创建

In [1]:
```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\subing\anaconda3\lib\site-packages (1.18.5)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import numpy as np
x1 = np.zeros((2,3))
print(x1)
```

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

numpy库数组属性

属性	描述
ndarray.ndim	数组轴的个数，也被称作秩
ndarray.shape	数组在每个维度上大小的整数元组
ndarray.size	数组元素的总个数
ndarray.dtype	数组元素的数据类型，dtype 类型可以用于创建数组中
ndarray.itemsize	数组中每个元素的字节大小
ndarray.data	包含实际数组元素的缓冲区地址
ndarray.flat	数组元素的迭代器

numpy库数组属性

- 数组属性

```
In [3]: x2 = np.ones((5,6))
        print(x2)
```

```
[[1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]]
```

```
In [4]: x2.ndim
```

```
Out[4]: 2
```

```
In [5]: x2.shape
```

```
Out[5]: (5, 6)
```

```
In [6]: x2.dtype
```

```
Out[6]: dtype('float64')
```

numpy库数组形态操作方法

- 数组是ndarray类型对象，可以采用<a>.<b>()方式调用一些方法。
- 改变数组基础形态的操作方法：例如改变和调换数组维度等。
- np.flatten()函数用于数组降维，相当于平铺数组中数据，该功能在矩阵运算及图像处理中用处很大。

方法	描述
ndarray.reshape(n,m)	不改变数组 ndarray，返回一个维度为(n,m)的数组
ndarray.resize(new_shape)	与 reshape()作用相同，直接修改数组 ndarray
ndarray.swapaxes(ax1, ax2)	将数组 n 个维度中任意两个维度进行调换
ndarray. flatten()	对数组进行降维，返回一个折叠后的一维数组
ndarray.ravel()	作用同 np.flatten()，但是返回数组的一个视图

numpy库数组形态操作方法

```
In [7]: x2.flatten()

Out[7]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
               1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
In [11]: x3 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], dtype=int)
         x4 = x3.reshape((3,4))
         print(x4)

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

```
In [12]: x3.resize((2,6))
         print(x3)

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

ndarray 类的索引和切片方法

- 数组切片得到的是原始数组的视图，所有修改都会直接反映到源数组。如果需要得到的ndarray 切片的一 份 副 本， 需 要 进 行 复 制 操 作， 比 如arange[5:8].copy()

方法	描述
x[i]	索引第 i 个元素
x[-i]	从后向前索引第 i 个元素
x[n:m]	默认步长为 1，从前往后索引，不包含 m
x[-m:-n]	默认步长为 1，从后往前索引，结束位置为 n

ndarray 类的索引和切片方法

```
In  [13]: x = np.random.rand(6,3)
          x[2]

Out[13]: array([0.71340748, 0.37877357, 0.02473368])

In  [14]: x[2:4]

Out[14]: array([[0.71340748, 0.37877357, 0.02473368],
                [0.84463945, 0.50083296, 0.76809101]])

In  [15]: x[-5:-2:2]

Out[15]: array([[0.14797886, 0.71228022, 0.0721306 ],
                [0.84463945, 0.50083296, 0.76809101]])
```

ndarray 类的算术运算函数

- 加减乘除等算术运算函数

函数	描述
np.add(x1, x2 [, y])	y = x1 + x2
np.subtract(x1, x2 [, y])	y = x1 - x2
np.multiply(x1, x2 [, y])	y = x1 * x2
np.divide(x1, x2 [, y])	y = x1 / x2
np floor_divide(x1, x2 [, y])	y = x1 // x2，返回值取整
np.negative(x [,y])	y = -x
np.power(x1, x2 [, y])	y = x1**x2
np.remainder(x1, x2 [, y])	y = x1 % x2

ndarray 类的算术运算函数

- 这些函数中，输出参数y 可选，如果没有指定，将创建并返回一个新的数组保存计算结果；如果指定参数，则将结果保存到参数中。例如，两个数组相加可以简单地写为a+b，而np.add(a,b,a)则表示a+=b。

In [4]:
```
import numpy as np
x = np.random.rand(3,2)
y = np.ones((3,2))
np.add(x,y,x)
print(x)
```

```
[[1.68169165 1.54105006]
 [1.66019779 1.07333519]
 [1.2191459  1.74021582]]
```

In [5]:
```
z = x + y
print(z)
```

```
[[2.68169165 2.54105006]
 [2.66019779 2.07333519]
 [2.2191459  2.74021582]]
```

ndarray 类的比较运算函数

- 比较运算函数：返回一个布尔数组，包含两个数组中对应元素值的比较结果

函数	符号描述
np. equal(x1, x2 [, y])	y = x1 == x2
np. not_equal(x1, x2 [, y])	y = x1 != x2
np. less(x1, x2, [, y])	y = x1 < x2
np. less_equal(x1, x2, [, y])	y = x1 <= x2
np. greater(x1, x2, [, y])	y = x1 > x2
np. greater_equal(x1, x2, [, y])	y = x1 >= x2
np.where(condition[x,y])	根据给出的条件判断输出 x 还是 y

ndarray 类的比较运算函数

- where()函数是三元表达式x if condition else y 的矢量版本

```
In  [6]: np.less(z, 2.5)

Out[6]: array([[False, False],
               [False,  True],
               [ True, False]])

In  [7]: np.less(z, [[2.5, 2.5], [2.5, 2.5], [2.5, 2.5]])

Out[7]: array([[False, False],
               [False,  True],
               [ True, False]])

In  [9]: np.where(z>2.5, 0, z)

Out[9]: array([[0.        , 0.        ],
               [0.        , 2.07333519],
               [2.2191459 , 0.        ]])
```

numpy库矩阵运算

- 矩阵乘法、转置、逆

```
import numpy as np
a1 = np.array([[1,2,3],[4,5,6]]) # a1为2*3矩阵
a2 = np.array([[1,2],[3,4],[5,6]]) # a2为3*2矩阵

print(a1.dot(a2))
b = np.dot(a1,a2)
print(b)

c = np.matmul(a1,a2)
print(c)
print(c.transpose())
```

```
[[22 28]
 [49 64]]
[[22 28]
 [49 64]]
[[22 28]
 [49 64]]
[[22 49]
 [28 64]]
```

```
import numpy.linalg as lg
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(lg.inv(a))

a = np.eye(3) # 3阶单位矩阵
print(lg.inv(a)) # 单位矩阵的逆为他本身

c = a.tolist()
print(c)
print(type(c))
```

```
[[ 3.15251974e+15 -6.30503948e+15  3.15251974e+15]
 [-6.30503948e+15  1.26100790e+16 -6.30503948e+15]
 [ 3.15251974e+15 -6.30503948e+15  3.15251974e+15]]
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
[[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
<class 'list'>
```

numpy库矩阵运算

- 乘法*

```
a1 = np.array([[1,2,3],[4,5,6]])
a2 = np.array([[3,2,1],[-1,-2,1]])
a3 = 3
print(a1*a2)
print(a3*a1)
```

```
[[  3   4   3]
 [ -4 -10   6]]
[[ 3  6  9]
 [12 15 18]]
```

numpy库其它运算函数

- numpy 库还包括三角运算函数、傅里叶变换、随机和概率分布、基本数值统计、位运算等非常丰富的功能

函数	描述
np.abs(x)	计算基于元素的整形，浮点或复数的绝对值。
np.sqrt(x)	计算每个元素的平方根
np.squre(x)	计算每个元素的平方
np.sign(x)	计算每个元素的符号：1(+)，0， -1(-)
np.ceil(x)	计算大于或等于每个元素的最小值
np.floor(x)	计算小于或等于每个元素的最大值
np.rint (x[, out])	圆整,取每个元素为最近的整数,保留数据类型
np.exp(x[, out])	计算每个元素指数值
np.log(x), np.log10(x), np.log2(x)	计算自然对数(e),基于 10,2 的对数,log(1 + x)

numpy库数据存储

- numpy能够读写磁盘上的文本数据或二进制数据
- 以二进制格式将数组保存到磁盘：np.save，默认情况下，数组是以未压缩的原始二进制格式保存在扩展名为.npy的文件中
- np.load从磁盘读取.npy文件

```
In [13]: np.save('C:/RUC/课程/PythonAI/课程课件/testz.npy',z)
         y = np.load('C:/RUC/课程/PythonAI/课程课件/testz.npy')
         print(y)
```

```
[[2.68169165 2.54105006]
 [2.66019779 2.07333519]
 [2.2191459  2.74021582]]
```

numpy库数据存储

- numpy.savez函数可以将多个数组保存到同一个文件中：第一个参数是文件名，其后的参数都是需要保存的数组
- 可以使用关键字参数为数组起一个名字，非关键字参数传递的数组会自动起名为arr_0, arr_1, ...
- 输出的是一个压缩文件(扩展名为npz)，其中每个文件都是一个save函数保存的npy文件，文件名对应于数组名
- load函数自动识别npz文件，并且返回一个类似于字典的对象，可以通过数组名作为关键字获取数组的内容

numpy库数据存储

In [14]:
```
np.savez('C:/RUC/课程/PythonAI/课程课件/testarray.npz',x,y,res_z=z)
Res = np.load('C:/RUC/课程/PythonAI/课程课件/testarray.npz')
print(Res['arr_0'])
```

```
[[1.68169165 1.54105006]
 [1.66019779 1.07333519]
 [1.2191459  1.74021582]]
```

In [15]:
```
print(Res['arr_1'])
```

```
[[2.68169165 2.54105006]
 [2.66019779 2.07333519]
 [2.2191459  2.74021582]]
```

In [16]:
```
print(Res['res_z'])
```

```
[[2.68169165 2.54105006]
 [2.66019779 2.07333519]
 [2.2191459  2.74021582]]
```

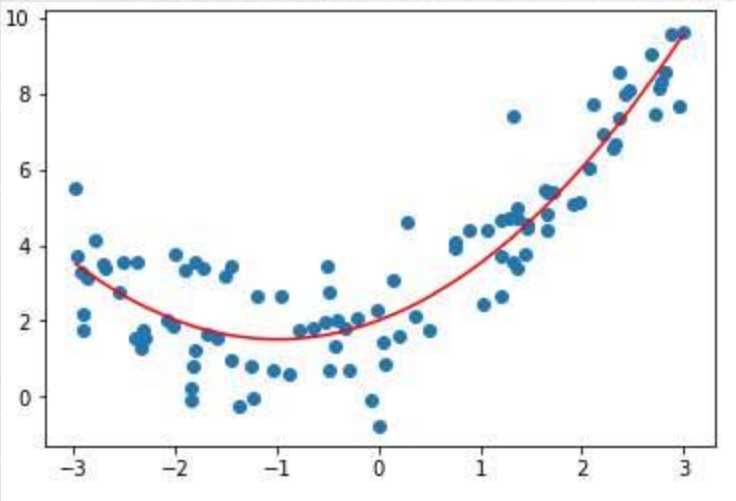
Python AI
Numpy与科学计算

提纲

- 第三方库安装
- numpy库
- **线性回归**

回归分析

- 回归分析是一种预测性的建模技术
  - 因变量（目标）和自变量（特征）之间的关系
  - 用于预测分析、时间序列模型以及发现变量之间的因果关系。
- 通常使用曲线/线来拟合数据点，目标是使数据点到曲线的距离差异最小。



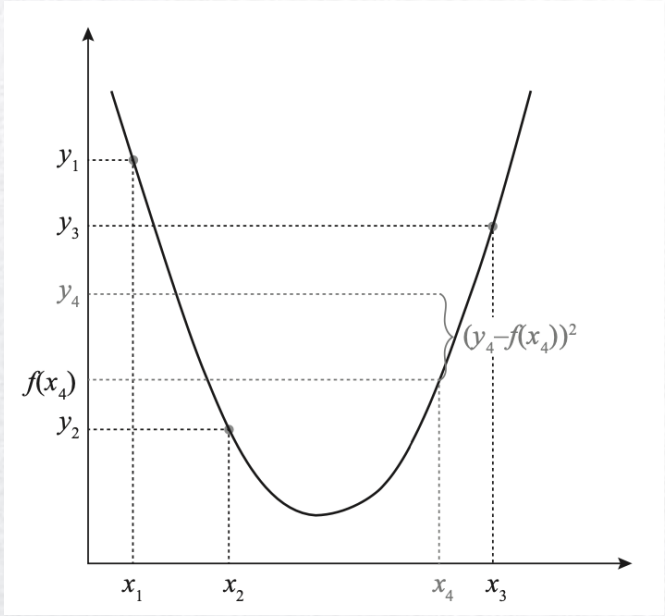
训练集/测试集

- 训练集
  - 给定样本的特征和目标值，用来训练或者拟合模型，获得模型参数。
- 测试集
  - 用来检验训练得到的模型的性能，评估模型的泛化能力。

- 给定数据集，首先需要做训练/测试集划分。

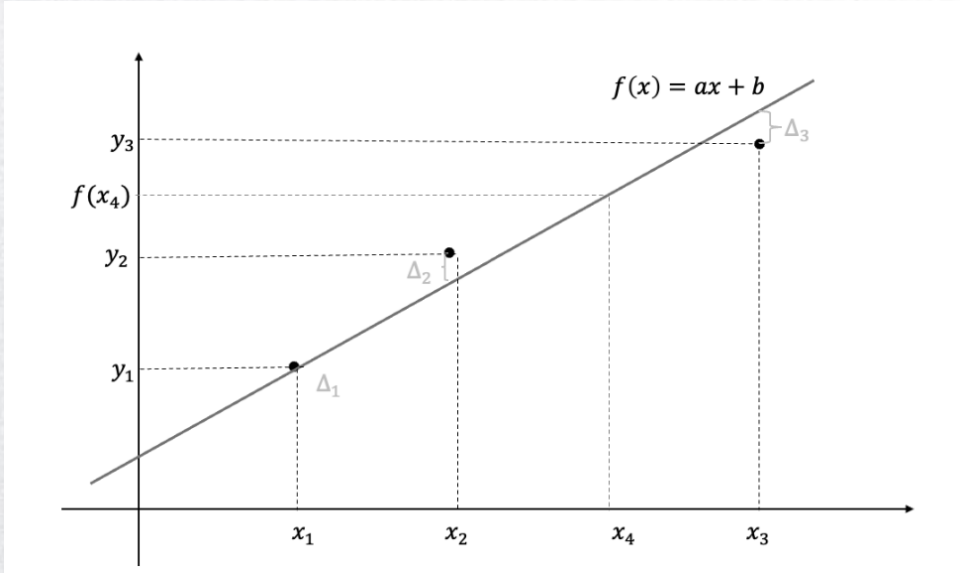
一元回归举例

- 在坐标纸上给定三个点$(x_1, y_1), (x_2, y_2), (x_3, y_3)$（训练集合），用平滑的曲线把这三个点连接起来
  - 回归模型：平滑的曲线$f(x)$
- 给定一个新的位置x_4（测试数据），在坐标纸上标出$f(x_4)$的值（对x_4点对应值进行预测）
  - 回归预测
- x_4处真实的值是y_4，因此预测误差为$\left(y_4 - f(x_4)\right)^2$
  - 回归误差



一元线性回归

- 在坐标纸上给定三个点的训练集合
  - $(x_1, y_1), (x_2, y_2), (x_3, y_3)$
- 并且规定$f(x)$为线性函数（直线）
  - $f(x) = \mathrm{w}x + b$
- 三个点，一条直线
  - 可能无法满足这条直线完美穿越所有的点（训练集合上的误差）
- 训练误差的计算
  - $\frac{1}{3}\left((f(x_1) - y_1)^2 + (f(x_2) - y_2)^2 + (f(x_3) - y_3)^2\right)$



- 模型训练要解决的问题：找到使得训练误差最小的参数组合(w, b)

损失函数

- 使用任何一组参数都可以得到一组预测值$\hat{y}$，需要一个标准来对预测结果进行度量：定量化一个目标函数式度量预测结果的好坏。
- MSE(mean square error)均方误差：线性模型应用于训练样本x_i，得到一组预测值$\hat{y}_i$，将预测值和真实值 y_i之间的平均的平方距离定义为损失函数：

$$L = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

- N为样本总数

损失函数

- 将线性预测函数式代入损失函数，将需要求解的参数w和b看做是损失函数L的自变量：

$$L(w,b)=\frac{1}{N}\sum_{i=1}^{N}(wx_i+b-y_i)^2$$

- 通过最小化损失函数求解最优参数：

$$(w^*,b^*)=\arg\min_{w,b}\frac{1}{N}\sum_{i=1}^{N}(wx_i+b-y_i)^2$$

优化

$$(w^*, b^*) = \arg\min_{w,b} \frac{1}{N} \sum_{i=1}^{N} (wx_i + b - y_i)^2$$

- 对w求导:

$$\frac{\partial L(w,b)}{\partial w} = \frac{1}{N} \sum_{i=1}^{N} 2(wx_i + b - y_i)x_i$$

优化

$$(w^*, b^*) = \arg\min_{w,b} \frac{1}{N} \sum_{i=1}^{N} (wx_i + b - y_i)^2$$

- 对b求导:

$$\frac{\partial L(w,b)}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} 2(wx_i + b - y_i)$$

优化：梯度下降法

- 1. 随机选择一组初始参数(w^0, b^0)
- 2. 对于可微函数，梯度方向时函数增长速度最快的方向，梯度的反方向是函数减少最快的方向。计算损失函数在当前参数点处的梯度。
- 3. 从当前参数点向梯度相反的方向移动，移动步长为:

$$w^{t+1} = w^t - lr\frac{\partial L(w,b)}{\partial w}(w^t)$$

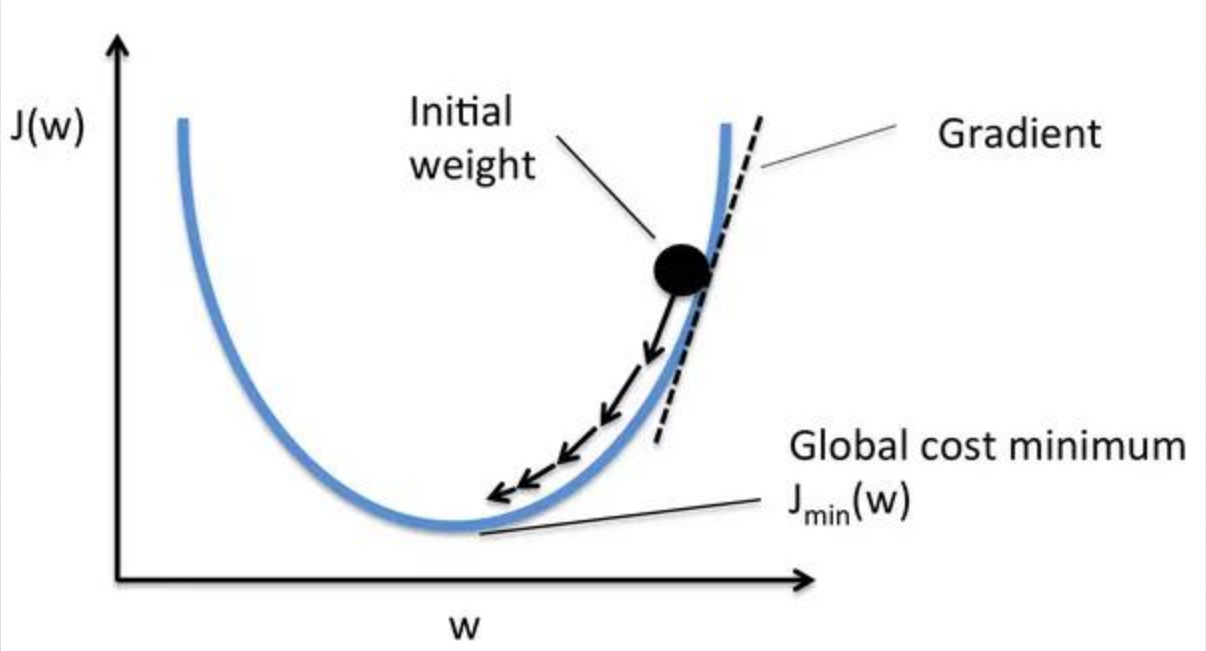
$$b^{t+1} = b^t - lr\frac{\partial L(w,b)}{\partial b}(b^t)$$

学习率

- 4. 循环迭代步骤 3， 直到前后两次迭代得到的(w^t, b^t)差值足够小，即参数基本不再变化，说明此时损失函数已经达到局部最小值。
- 5. 输出(w^t, b^t)，即为使得损失函数最小时的参数取值 。

梯度下降法

- 一元线性回归



示例：梯度下降法

```
class LinearRegression(object):

    def __init__(self, learning_rate=0.01, max_iter=100, seed=None):
        '''
        一元线性回归类的构造函数：
        参数 学习率：learning_rate
        参数 最大迭代次数：max_iter
        参数 seed：产生随机数的种子
        从正态分布中采样w和b的初始值
        '''
        np.random.seed(seed)
        self.lr = learning_rate
        self.max_iter = max_iter
        self.w = np.random.normal(1, 0.1)
        self.b = np.random.normal(1, 0.1)
        self.loss_arr = []
```

示例：梯度下降法

```
def fit(self, x, y):
    '''
    类的方法：训练函数
    参数 自变量：x
    参数 因变量：y
    返回每一次迭代后的损失函数
    '''
    for i in range(self.max_iter):
        self.__train_step(x,y)
        y_pred = self.predict(x)
        self.loss_arr.append(self.loss(y, y_pred))

def __f(self, x, w, b):
    '''
    类的方法：计算一元线性回归函数在x处的值
    '''
    return x * w + b
```

```
def predict(self, x):
    '''
    类的方法：预测函数
    参数：自变量：x
    返回：对x的回归值
    '''
    y_pred = self.__f(x, self.w, self.b)
    return y_pred

def loss(self, y_true, y_pred):
    '''
    类的方法：计算损失
    参数 真实因变量：y_true
    参数 预测因变量：y_pred
    返回：MSE损失
    '''
    return np.mean((y_true - y_pred)**2)
```

示例：梯度下降法

```
def __calc_gradient(self, x, y):
    '''
    类的方法：分别计算对w和b的梯度
    '''
    d_w = np.mean(2* (x * self.w + self.b - y) * x)
    d_b = np.mean(2*(x * self.w + self.b - y))
    return d_w, d_b

def __train_step(self,x,y):
    '''
    类的方法：单步迭代，即一次迭代中对梯度进行更新
    '''
    d_w, d_b = self.__calc_gradient(x,y)
    self.w = self.w - self.lr * d_w
    self.b = self.b - self.lr * d_b
    return self.w, self.b
```

示例：梯度下降法

In [36]:

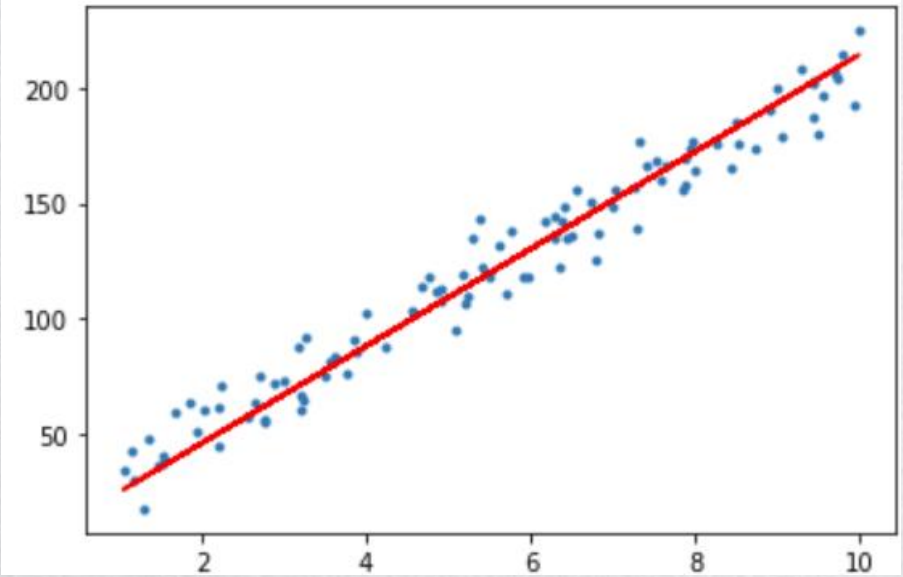
```
import numpy as np
import matplotlib.pyplot as plt

def show_data(x, y, w=None, b=None):
    plt.scatter(x, y, marker='.')
    if w is not None and b is not None:
        plt.plot(x, w*x+b, c='red')
    plt.show()

# data generation
np.random.seed(272)
data_size = 100
x = np.random.uniform(low=1.0, high=10.0, size=data_size)
y = x * 20 + 10 + np.random.normal(loc=0.0, scale=10.0, size=data_size)
```

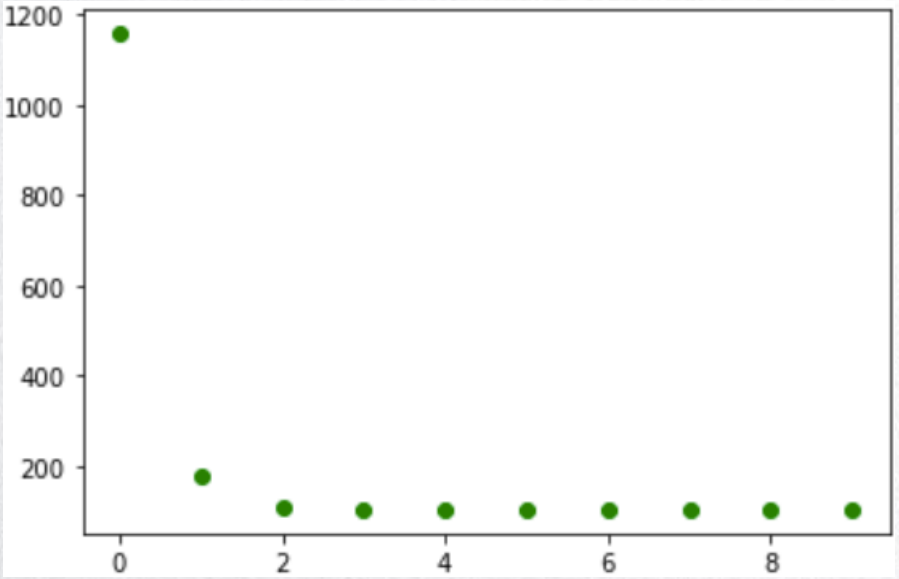
示例：梯度下降法

```
# train / test split
shuffled_index = np.random.permutation(data_size)
x = x[shuffled_index]
y = y[shuffled_index]
split_index = int(data_size * 0.7)
x_train = x[:split_index]
y_train = y[:split_index]
x_test = x[split_index:]
y_test = y[split_index:]

# train the liner regression model
regr = LinearRegression(learning_rate=0.01, max_iter=10, seed=314)
regr.fit(x_train, y_train)
print('w: \t{:.3}'.format(regr.w))
print('b: \t{:.3}'.format(regr.b))
show_data(x, y, regr.w, regr.b)

# plot the evolution of cost
plt.scatter(np.arange(len(regr.loss_arr)), regr.loss_arr, marker='o', c='green')
plt.show()
```

示例：梯度下降法

```
w:      21.0
b:      4.41
```





思考

- 多元线性回归：
  - 自变量x是向量，因变量y是数字
  - 自变量x是向量，因变量y也是向量

- 怎样用梯度下降法训练多元线性回归模型？

谢谢！