面向数据科学的数据分析教程

李俊

2019 年 03 月 20 日

目录

1

Python 中的科学计算工具

Python 在解决数据科学任务和挑战方面继续处于领先地位，本章主要介绍 Python 中几个最基础、也是最重要科学技计算包。

NumPy

NumPy 是科学应用程序库的主要软件包之一，用于处理大型多维数组和矩阵，它大量的高级数学函数集合和实现方法使得这些对象执行操作成为可能。

官网：http://www.numpy.org/

Pandas

Pandas 是一个 Python 库，提供高级的数据结构和各种各样的分析工具。这个软件包的主要特点是能够将相当复杂的数据操作转换为一两个命令。Pandas包含许多用于分组、过滤和组合数据的内置方法，以及时间序列功能。

官网：https://pandas.pydata.org/

Matplotlib

Matplotlib 是一个用于创建二维图和图形的底层库。藉由它的帮助，你可以构建各种不同的图标，从直方图和散点图到费笛卡尔坐标图。

Matplotlib 就是 MATLAB+Plot+Library，即模仿 MATLAB 的绘图库，其绘图风格与 MATLAB 类似。

官网：https://matplotlib.org/index.html

Seaborn

Seaborn 本质上是一个基于 matplotlib 库的高级 API。它包含更适合处理图表的默认设置。此外，还有丰富的可视化库，包括一些复杂类型，如时间序列、联合分布图

官网：https://seaborn.pydata.org/

Scikit-learn

Scikit-learn 是 Python 中简单高效的数据挖掘和数据分析工具，建立在 NumPy, SciPy 和 matplotlib 的基础上，为许多标准的机器学习和数据挖掘任务提供算法，如聚类、回归、分类、降维和模型选择。

官网：http://scikit-learn.org/stable/

中文文档：http://sklearn.apachecn.org/cn/0.19.0/

SciPy

SciPy 基于 NumPy，其功能也因此得到了扩展。SciPy 主数据结构又是一个多维数组，由 Numpy 实现,包含了帮助解决线性代数、概率论、积分计算和许多其他任务的工具。

官网：https://scipy.org/scipylib/

Plotly

Plotly 是一个动态绘图模块，它可以让你轻松构建复杂的图形。该软件包适用于交互式 Web 应用程，可实现轮廓图、三元图和三维图等视觉效果。

官网：https://plot.ly/python/

参考文章

2018：数据科学20个最好的Python库

1.1 Python

Python语言是一种解释型、面向对象、动态数据类型的高级程序设计语言,如今已成为绝大部分数据分析师的首选数据分析语言。其设计的核心理念是代码的易读性，以及允许编程者通过若干行代码轻松表达想法创意。

1.1.1 版本

本教程所有代码采用 Python3.6 进行编写，在 Python 可通过以下命令查看版本：

[1]:
```
import sys
sys.version
```

[1]:
```
'3.6.8 |Anaconda, Inc.| (default, Dec 30 2018, 18:50:55) [MSC v.1915␣
↪64 bit (AMD64)]'
```

在 Jupyter Notebook 中通过!python --version命令查看版本。

[2]:
```
!python --version
```

```
Python 3.6.8 :: Anaconda, Inc.
```

1.1.2 语法

缩进

锁紧是表达代码间包含和层次关系的唯一手段，在 Python 中通过 4 个空格或 1 个 TAB 键来表示。

注释

注释是不被程序执行的辅助性说明信息。在 Python 中单注释分为单行和多行，单行注释以#开头，其后内容为注释；多行注释以'''开头和结尾，中间内容为注释。

[3]:
```
# 这是一个单行注释

'''
这是一个多行注释
这是一个多行注释
'''
print('注释')
```

[3]:
```
'\n这是一个多行注释\n这是一个多行注释\n'
```

变量

变量是用来保存和表示数据的占位符号。在 Python 中通过等号=用向变量赋值或修改值。变量名是大小写字母、数字、下划线_和汉字等字符的组合，对大小写敏感，首字母不能是数字，也不能与保留字相同。

[4]:
```
# 向变量x赋值为1
x = 1
print('x =',x)

# 修改变量x的赋值为2
x = 2
print('x =',x)
```

```
x = 1
x = 2
```

保留字

保留字是被编程语言内部定义并保留使用的标识符。在 Python 中有 33 个保留字，保留字是编程语言的基本单词，对大小写敏感。

and	as	assert	break	class	continue
def	elif	else	except	finally	for
from	if	import	in	is	lambda
not	or	pass	raise	return	try
while	with	yield	del	global	nonlocal
True	False	None			

1.1.3 操作符

操作符是完成运算的一种符号体系

运算操作符

操作符	描述
x + y	加法运算
x - y	减法运算
x * y	乘法运算
x / y	除法运算
x // y	整除运算
x % y	取模(余数)运算
- y	负数运算
x ** y	幂运算，y 为小数时为开方运算

赋值操作符

操作符	描述
x += y	即 x = x + y
x -= y	即 x = x - y
x *= y	即 x = x * y
x /= y	即 x = x / y
x //= y	即 x = x // y
x %= y	即 x = x % y
x **= y	即 x = x ** y

比较运算符

比较运算结果为 True 或 False。

操作符	描述
==	等于
！=	不等于
<>	不等于，与！= 类似
>	大与
<	小于
>+	大于等于
<=	小于等于

1.1.4 数据类型

数据类型是供计算机程序理解的数据形式，在 Python 中数据类型被分为基本数据类型和复合数据类型。

- 基本数据类型：包括整数、浮点数、复数、布尔数值和空值。
- 复合数据类型：包括字符串、列表、元组、字典和集合。

整数（int）

Python 中的整数与数学中整数的概念一致，可正可负，没有取值范围限制，在 Python 中有 4 种表达形式。

表达形式	格式
十进制	-100,0,100
二进制	以0b或0B开头：0b01,-0B01
八进制	以0o或0O开头：0o17,-0O17
十六进制	以0x或0X开头：0x0f, -0X0f

整数虽然有 4 种表达形式，但输出格式都是十进制。

[5]:
```
print("二进制数 0b01 的输出格式是：",0b01)
print("八进制数 0o17 的输出格式是：",0o17)
print("十六进制数 0x0f 的输出格式是：",0x0f)
```

```
二进制数 0b01 的输出格式是： 1
八进制数 0o17 的输出格式是： 15
十六进制数 0x0f 的输出格式是： 15
```

浮点数（float）

Python 中的浮点数与数学中实数的概念一致，指带有小数点及小数的数字。浮点数取值范围和小数精度都存在限制，但常规计算可忽略。取值范围数量级约-10^{308}至10^{308}，精度数量级10^{16}。

在 Python 中浮点数有两种表达形式：一种用小数点+数字形式表示；一种用科学计数法表示，使用字母e或E作为幂的符号，以10为基数，$< a > e < b >$表示$a * 10^b$

[6]:
```
print('浮点数 110e-3 值为：',110e-3)
print('浮点数 5.2E2 值为：',5.2E2)
```

```
浮点数 110e-3 值为： 0.11
浮点数 5.2E2 值为： 520.0
```

复数（plural）

Python 中的复数与数学中复数的概念一致，记 a+b*j 为复数，其中 a 是实部，b 是虚部。

[7]:
```
x = 1 + 2j
print('复数 1 + 2j 的实部是：', x.real)
print('复数 1 + 2j 的虚部是：', x.imag)
```

```
复数 1 + 2j 的实部是： 1.0
复数 1 + 2j 的虚部是： 2.0
```

布尔数值（bool）

Python 中的布尔数值与数学中布尔代数的概念一致，分为`True`和`False`。利用布尔数值可进行逻辑运算。

[8]:
```
x = True
y = False
```

空值（None）

空值是 Python 中一种特殊的数据类型，用 None表示。None表示一个空对象，不能理解为0。

[9]:
```
x = None
x == 0
```

[9]:
```
False
```

序列（sequence）

序列是具有先后关系的一组元素，各元素类型可以不同。元素间通过序号引导，通过下标访问序列的特定元素。字符串、列表和元组类型都属于序列类型。

序号分为两种：正向递增序号和正向递减序号，对应关系如下表：

正向递增序号	0	1	2	⋯	n-1
序列	1	2	3	⋯	n
反向递减序号	-n	-n+1	-n+2	⋯	-1

利用序号可以对序列进行索引和切片操作：

- 索引指通过序号选取序列中特定元素的操作，通过sequence[index]使用；
- 切片指从序列获取多个元素的操作，通过sequence[start_index:end_index:step]使用，start_index表示开始索引位置，默认为0；end_index表示结束索引位置，默认为n-1；step表示步长，默认为1。切片操作前闭后开，不包括sequence[end_index]。

[10]:
```
x = [1, 2, 3, 4, 5]

print('x[1] = ',x[1])
print('x[-1] =',x[-1])
print('x[1:4:2] =',x[1:4:2])
print('x[::-1] =',x[::-1])
```

```
x[1] =  2
x[-1] = 5
x[1:4:2] = [2, 4]
x[::-1] = [5, 4, 3, 2, 1]
```

字符串、列表和元组类型作为序列的子类，都拥有序列的通用函数和方法：

函数和方法	描述
len(x)	返回序列 x 的长度
min(x)	返回序列 x 的最小元素
max(x)	返回序列 x 的最大元素
x.index(a)或x.index(a,i,j)	返回序列 x 从 i 开始到 j 位置中第一次出现元素 a 的位置
x.count(a)	返回序列 x 中出现元素 a 的总次数

字符串

字符串是由 0 个或多个字符组成的有序字符序列，由一对单引号''或一对双引号""表示。

[11]:
```
print('我是一个字符串')
print("我是另一个字符串")
print("'我还是一个字符串'")
```

```
我是一个字符串
我是另一个字符串
'我还是一个字符串'
```

如果现在字符串中输入一个单引号'或双引号"，需要通过转义符\表达

[12]:
```
print('这是一个单引号（\'）')
print('这是一个双引号（\"）')
```

```
这是一个单引号（'）
这是一个双引号（"）
```

转义符和字母进行组合可以表达出一些字符串无法表达的含义

- \b: 后退
- \n: 换行，即光标移动到下行行首
- \r: 回车，即贯标移动到本行行首

字符串继承序列类型的函数和方法，但也具有独有的函数和方法：

函数和方法	描述
str(x)	返回任意类型x所对应的字符串类型，如`str(123) = "123"`
hex(x) 或 oct(x)	返回整数x的十六进制或八进制小写形式字符串，如`hex(425) = "0x1a9"`
chr(x)	返回 Unicode 编码x对应的字符，如`chr(1000) = 'Ϩ'`
ord(x)	返回字符x对应的 Unicode 编码，如`ord('Ϩ') = 1000`
str.lower() 或 str.upper()	返回字符串的副本，全部字符小写/大写,如`'AbCd'.lower() = 'abcd'`
str.split(sep=None)	返回一个列表, 由str根据sep被分隔的部分组, 如`'a,b,c'.split(',') = ['a','b','c']`
str.count(sub)	返回子串sub在str中出现的次数，如`'abca'.count('a') = 2`
str.replace(old, new)	返回字符串str副本，所有old子串被替换为new，如`'abc'.replace('a', 'ab') = 'abbc'`

列表（list）

列表是一种序列类型，创建后可以随意被修改。使用方括号[]或list()创建，元素间用逗号,分隔。列表中各元素类型可以不同，无长度限制。

[13]:
```
x = ['abc', 123]
print(x)

y = list('123')
print(y)
```

```
['abc', 123]
['1', '2', '3']
```

列表继承序列类型的函数和方法，但也具有独有的函数和方法：

函数和方法	描述
list.append(x)	在list最后增加一个元素x
list.clear()	删除list中所有元素
list.copy()	复制list，并生成一个新列表
list.inerst(i,x)	在list的第i位置增加元素x
list.pop(i)	返回list中第i位置元素，并在list中删除该元素
list.remove(x)	删除list出现的第一个元素x
list.reverse()	颠倒list

元组（tuple）

元组是一种序列类型，一旦创建就不能被修。使用小括号()或tuple()创建，元素间用逗号,分隔。可以使用或不使用小括号

[14]:
```
x = 'abc',123
x
```

[14]:
```
('abc', 123)
```

元组继承了序列类型的全部通用操作，因为创建后不能修改，因此没有特殊操作。

字典（dict）

字典是一种储存键值对的数据类型，键值对之间无序。通过大括号{}和dict()创建，键值对用冒号:表示。在字典变量中，通过键获得值。

键（keys）可以看做索引的扩展，值（values）储存数据。

[15]:
```
x = {'a':1, 'b':2}
print('x =',x)
print("x['a'] =",x['a'])
```

```
x = {'a': 1, 'b': 2}
x['a'] = 1
```

字典类型操作函数和方法

函数和方法	描述
dict.keys()	返回dict中所有键信息
dict.values()	返回dict中所有值信息
dict.items()	返回dict中所有键值对信息
dict.get(k, x)	若键k存在，则返回对应的值，否则返回x值
dict.pop(k, x)	若键k存在，则取出对应的值，否则返回x值
dict.popitem()	从dict随机取出一个键值对，以元组形式返回
dict.clear()	删除dict所有键值对
len(dict)	返回dice中元素个数

集合（set）

Python 的集合类型与数学中的集合概念一致，是多个元素的无序组合。元素与元素间无序，不存在相同元素。集合中的元素不可更改，不能使可变数据类型。

集合通过大括号{}或set()创建，元素间用逗号,分割，创建空集合必须使用set()方法。

[16]:
```
x = {'a', 1, 10, 'a'}
x
```

[16]:
```
{1, 10, 'a'}
```

1.1.5 函数

函数（function）

函数是一段具有特定功能的、可重用的语句组，是一种功能的抽象。

在 Python 中通过def构建函数，格式如下：

```
def <函数名>(<参数(0或多个)>):
    <函数体>
    return <返回值>
```

[17]:
```
# 计算斐波那契数列
def f(n):
    if n==1:
```

(下页继续)

(续上页)

```
        return 1
    elif n==2:
        return 1
    else:
        return f(n-1)+f(n-2)

# 第10个斐波那契数
f(10)
```

[17]:
```
55
```

1.1.6 参考资料

北京理工大学：Python语言程序设计

菜鸟教程：Python3教程

1.2 Jupyter Notebook

1.3 Numpy

Numpy 是 Python 中用于科学计算的基本包。它是一个 Python 库，提供多维数组对象、各种派生对象(如屏蔽数组和矩阵)，以及用于数组快速操作的各种例程，包括数学、逻辑、形状操作、排序、选择、i/o、离散傅立叶变换、基本线性代数、基本统计操作、随机模拟等等。

1.3.1 版本

[1]:
```
import numpy
numpy.__version__
```

[1]:
```
'1.15.4'
```

1.3.2 别名

遵循传统，使用np作为别名导入 NumPy：

[2]:
```
import numpy as np
```

1.3.3 数组

Numpy 包的核心是ndarray对象，封装了同类数据类型的 n 维数组，通常用别名array来表示。

属性

每个Numpy数组都拥有如下属性：

- nidm：数组维度
- shape：维度大小
- size：数组大小
- dtype：数据类型
- itemsize：元素字节大小, 以bytes为单位
- nbytes：数组字节大小, 以bytes为单位

[3]:
```
# 创建一个3×3、由[0,10)均匀分布随机整数组成的数值
x = (np.random.randint(0, 10, (3, 3)))
print(x)
print('\nx.ndim:',x.ndim)
print('x.shape:',x.shape)
print('x.size:',x.size)
print('x.dtype:',x.dtype)
print('x.itemsize:',x.itemsize)
print('x.nbytes:',x.nbytes)
```

```
[[3 4 9]
 [0 0 6]
 [4 4 3]]

x.ndim: 2
x.shape: (3, 3)
x.size: 9
```

(下页继续)

(续上页)

```
x.dtype: int32
x.itemsize: 4
x.nbytes: 36
```

利用列表(list)生成数组

利用nd.array从 Pyhon 列表创建数值:

[4]:
```
x = np.array([1,2,3,4])
x
```

[4]:
```
array([1, 2, 3, 4])
```

不同于 Python 列表，Numpy 要求数组必须包含同一类型的数据。如果类型不匹配，NumPy 将会向上转换（如果可行）。

[5]:
```
x = np.array([1.0, 2, 3.0, 4])
x
```

[5]:
```
array([1., 2., 3., 4.])
```

如果希望明确设置数组的数据类型，可以用dtype变量:

[6]:
```
x = np.array([1, 2, 3, 4], dtype=np.float32)
x
```

[6]:
```
array([1., 2., 3., 4.], dtype=float32)
```

构建多维数组:

[7]:
```
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
x
```

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

利用Numpy内置方法创建数值

np.zeros():创建全 0 数值

[8]:
```
# 创建一个长度为5的全0数组
x = np.zeros(5)
x
```

[8]:
```
array([0., 0., 0., 0., 0.])
```

np.ones():创建全 1 数组

[9]:
```
# 创建一个3*3的q全1矩阵
x = np.ones((3,3), dtype=float)
x
```

[9]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

np.eye():创建一个单位矩阵

[10]:
```
# 创建一个3*3的单位矩阵
x = np.eye(3)
x
```

[10]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

np.full(shape, fill_value): 创建一个维数为shape、值为fill_value的数组

[11]:
```
# 创建一个维数为3*3、值为2.0的浮点数型数组
x = np.full((3,3), fill_value=2.0)
x
```

[11]:
```
array([[2., 2., 2.],
       [2., 2., 2.],
       [2., 2., 2.]])
```

序列数组:

- np.arange(start, stop, step)：创建一个从start到stop,步长为step的序列数组。
- np.linspace(start, stop, num=50)：创建一个从start到stop,元素个数为num的序列数组。

[12]:
```
# 创建一个从0到10,步长为2的序列数组
x = np.arange(0, 10, 2)
x
```

[12]:
```
array([0, 2, 4, 6, 8])
```

[13]:
```
# 创建一个从0到10,元素个数为5的序列数组
x = np.linspace(0, 10, 5)
x
```

[13]:
```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

随机分布数组：

- np.random.random()：创建一个由 0 ~ 1 均匀分布生成的随机数组成的数组
- np.random.normal(loc=0.0, scale=1.0, size=None)：创建一个由均值为loc、方差为scale的正态分布生成的随机数组成的数组
- np.random.randint(low, high=None, size=None)：创建一个由low~high均匀分布生成的随机整数组成的数值

[14]:
```
# 创建一个3×3、由0~1均匀分布随机数组成的数组
x = np.random.random((3, 3))
x
```

[14]:
```
array([[0.36002842, 0.08519123, 0.4640263 ],
       [0.54155948, 0.43645734, 0.44322506],
       [0.30741928, 0.2057965 , 0.21929336]])
```

[15]:
```
# 创建一个3×3、由均值为0、方差为1的正态分布随机数组成的数组
x = np.random.normal(0, 1, (3, 3))
x
```

[15]:
```
array([[-0.35800332, -0.36553495, -0.62851015],
       [-2.01214244,  0.42221338,  0.25461076],
       [ 0.64228871,  0.28348644,  0.69593449]])
```

[16]:
```
# 创建一个3×3、由[0,10)均匀分布随机整数组成的数值
x = (np.random.randint(0, 10, (3, 3)))
x
```

[16]:
```
array([[3, 6, 0],
       [3, 2, 3],
       [0, 9, 2]])
```

1.3.4 数组操作

基本运算

数组上的运算与 Python 类似，支持 Python 原生的算术运算符，标准的加、减、乘、除都可以使用。

[17]:
```
x = np.arange(4)
print("x =", x)
print("x + 5 =", x + 5)
print("x * 2 =", x * 2)
print("x / 2 =", x / 2)
print("x // 2 =", x // 2)
print("-x = ", -x)
print("x ** 2 = ", x ** 2)
print("x % 2 = ", x % 2)
```

```
x = [0 1 2 3]
x + 5 = [5 6 7 8]
x * 2 = [0 2 4 6]
x / 2 = [0.  0.5 1.  1.5]
x // 2 = [0 0 1 1]
-x =  [ 0 -1 -2 -3]
x ** 2 =  [0 1 4 9]
x % 2 =  [0 1 0 1]
```

运算符与通用函数的对应关系如下表所示：

运算符	对应的通用函数	描述
+	np.add	加法运算
-	np.subtract	减法运算
-	np.negative	负数运算
*	np.multipy	乘法运算
/	np.divide	除法运算
//	np.floor_divide	取商除法运算
**	np.power	指数运算
&	np.mod	取模(余数)除法运算

与许多矩阵语言不同，乘法运算符*在 Numpy 数组中按元素操作，矩阵运算可以使用@操作符(在python 3.5中)或dot函数执行。

[18]:
```
A = np.array( [[1,1],
               [0,1]] )
B = np.array( [[2,0],
               [3,4]] )
# 元素操作
print('A * B =\n',A * B)

# 矩阵运算
print('\nA @ B =\n',A @ B)

# 另一种矩阵运算
print('\nA.dot(B) =\n', A.dot(B))
```

```
A * B =
 [[2 0]
 [0 4]]

A @ B =
 [[5 4]
 [3 4]]

A.dot(B) =
 [[5 4]
```

(下页继续)

(续上页)

```
 [3 4]]
```

索引与切片

[19]:
```
# 创建一个3×3、由[0,10)均匀分布随机整数组成的数值
x = (np.random.randint(0, 10, (3, 3)))
print(x)

# 索引
print('\nx[1,1] =',x[1,1])

# 切片
print('x[1:,1:] =\n',x[1:,1:])
```

```
[[5 0 5]
 [1 9 6]
 [1 5 1]]

x[1,1] = 9
x[1:,1:] =
 [[9 6]
 [5 1]]
```

排序

- np.sort(a, axis=1)：a表示要排序的数值，axis=0表示按列排序(axis=1表示按行排序)

[20]:
```
# 创建一个4×6、由[0,10)均匀分布随机整数组成的数值
x = (np.random.randint(0, 10, (4, 6)))
print('x =\n',x)

# 按列排序
print('\nnp.sort(x, axis=0) =\n',np.sort(x, axis=0))
```

(下页继续)

(续上页)

```
# 按列排序
print('\nnp.sort(x, axis=1) =\n',np.sort(x, axis=1))
```

```
x =
 [[7 9 6 2 1 2]
 [9 3 4 5 8 7]
 [4 6 8 1 6 7]
 [5 7 8 6 1 2]]

np.sort(x, axis=0) =
 [[4 3 4 1 1 2]
 [5 6 6 2 1 2]
 [7 7 8 5 6 7]
 [9 9 8 6 8 7]]

np.sort(x, axis=1) =
 [[1 2 2 6 7 9]
 [3 4 5 7 8 9]
 [1 4 6 6 7 8]
 [1 2 5 6 7 8]]
```

变形

- reshape()方法

[21]:
```
# 创建一个3×3、由[0,10)均匀分布随机整数组成的数值
x = (np.random.randint(0, 10, (3, 3)))
print('x =\n',x)
print('\n通过变形获得的行向量:\n', x.reshape((1, 9)))
print('通过变形获得的列向量:\n', x.reshape((9, 1)))
```

```
x =
 [[9 3 5]
 [8 7 7]
 [2 7 2]]
```

(下页继续)

(续上页)

```
通过变形获得的行向量:
 [[9 3 5 8 7 7 2 7 2]]
通过变形获得的列向量:
 [[9]
 [3]
 [5]
 [8]
 [7]
 [7]
 [2]
 [7]
 [2]]
```

转置

[22]:
```
x = (np.random.randint(0, 10, (2, 3)))
print('x =\n',x)
print('\nx.T =\n',x.T)
```

```
x =
 [[0 4 0]
 [5 8 6]]

x.T =
 [[0 5]
 [4 8]
 [0 6]]
```

拼接

- np.concatenate((a1, a2, ...), axis=0):按垂直(水平axis=1)拼接数组元组(a1, a2, ...)或数值列表[a1, a2, ...]
- np.vstack((a1, a2, ...)): 按垂直方向拼接数组元组(a1, a2, ...)或数值列表[a1, a2, ...]

- np.hstack((a1, a2, ...)): 按水平方向拼接数组元组(a1, a2, ...)或数值列表[a1, a2, ...]

[23]:
```
x = np.array([[1,2,3,4],
                  [1,2,3,4]])
x
```

[23]:
```
array([[1, 2, 3, 4],
       [1, 2, 3, 4]])
```

[24]:
```
# 按垂直方向拼接数组x和x
print('通过np.concatenate()方法拼接:\n',np.concatenate((x,x)))
print('通过np.vstack()方法拼接:\n',np.vstack((x,x)))
```

```
通过np.concatenate()方法拼接:
 [[1 2 3 4]
 [1 2 3 4]
 [1 2 3 4]
 [1 2 3 4]]
通过np.vstack()方法拼接:
 [[1 2 3 4]
 [1 2 3 4]
 [1 2 3 4]
 [1 2 3 4]]
```

[25]:
```
# 按水平方向拼接数值x和x
print('通过np.concatenate()方法拼接:\n',np.concatenate((x,x),axis=1))
print('通过np.vstack()方法拼接:\n',np.hstack((x,x)))
```

```
通过np.concatenate()方法拼接:
 [[1 2 3 4 1 2 3 4]
 [1 2 3 4 1 2 3 4]]
通过np.vstack()方法拼接:
 [[1 2 3 4 1 2 3 4]
 [1 2 3 4 1 2 3 4]]
```

拆分

- np.split(ary, indices_or_sections, axis=0): ary为要拆分的数值, indices_or_sections为拆分节点序列, axis=0表示按竖直方向拆分(axis=1表示按竖直方向拆分)
- np.hsplit(): 按水平方向拆分
- np.vsplit(): 按竖直方向拆分

一维数组:

[26]:
```
x = np.arange(9)
print(x)
print(np.split(x,3))
print(np.split(x,[2,5,7]))
```

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

二维数组:

[27]:
```
# 按竖直方向拆分
y = (np.random.randint(0, 10, (3, 3)))
print('通过np.split()方法拆分:\n',np.split(y, 3))
print('通过np.hsplit()方法拆分:\n',np.vsplit(y, 3))
```

```
通过np.split()方法拆分:
 [array([[3, 4, 1]]), array([[9, 7, 0]]), array([[0, 0, 8]])]
通过np.hsplit()方法拆分:
 [array([[3, 4, 1]]), array([[9, 7, 0]]), array([[0, 0, 8]])]
```

[28]:
```
# 按水平方向拆分
print('通过np.split()方法拆分:\n',np.split(y, 3, axis=1))
print('通过np.hsplit()方法拆分:\n',np.hsplit(y, 3))
```

```
通过np.split()方法拆分:
 [array([[3],
        [9],
        [0]]), array([[4],
```

(下页继续)

(续上页)

```
       [7],
       [0]]), array([[1],
       [0],
       [8]])]
通过np.hsplit()方法拆分:
 [array([[3],
       [9],
       [0]]), array([[4],
       [7],
       [0]]), array([[1],
       [0],
       [8]])]
```

1.3.5 通用函数

Numpy 提供熟悉的数学函数,例如sin,cos和exp等,在 NumPy 中,这些被称为“通用函数”(ufunc)。

[29]:
```
x = np.arange(3)
print('x =',x)
print('np.exp(x) =',np.exp(x))
print('np.sqrt(x) =',np.sqrt(x))
print('np.add(x) =',np.add(x,x))
```

```
x = [0 1 2]
np.exp(x) = [1.         2.71828183 7.3890561 ]
np.sqrt(x) = [0.         1.         1.41421356]
np.add(x) = [0 2 4]
```

Numpy 提供了大量的通用函数，详细列表请点击这里

1.3.6 广播

广播是一种强有力的机制，通过它 NumPy 可以使不同大小的矩阵在一起进行数学计算。

[30]:
```
a = np.array([[ 0.0, 0.0, 0.0],
              [10.0,10.0,10.0],
```

(下页继续)

(续上页)

```
                [20.0,20.0,20.0],
                [30.0,30.0,30.0]])
b = np.array([1.0,2.0,3.0])
a + b
```

[30]:
```
array([[ 1.,  2.,  3.],
       [11., 12., 13.],
       [21., 22., 23.],
       [31., 32., 33.]])
```

想了解更多细节可以点击这里

1.3.7 参考资料

- Numpy 用户指南

[ ]:
```

```

1.4 Pandas

[ ]:
```

```

1.5 Matplotlib

1.6 Seaborn

1.7 Scikit-learn

2

数据预处理

2.1 数据清洗

3

线性回归

3.1 线性回归综述

虽然机器学习中千姿百态的模型让人眼花缭乱，它但究其本原，它们都来源于最原始的线性回归（linear regression）。

在一个线性回归模型中，通常将需要关注或预测的变量叫做输出变量（或因变量），而用来解释因变量变化的变量叫做输入变量（或自变量）。线性回归研究的输入变量与输出变量的关系，一般假设输出变量是若干输入变量的线性组合，再根据这一关系求解线性组合中的最优系数，就能运用所求得的模型进行预测或估计。

为了方便读者理解，我们先从最简单的一元线性回归模型讲起。

3.1.1 基本概念

我们先来了解一霞接下来将要用到的概念：

- **x**表示第输入变量，也称作输入特征（feature），即输入数据；
- **y**表示输出变量，也称作输入特征（feature），即输入数据；
- x_i表示输入变量的第i个标签，即**x**= $(x_1, x_2, ..., x_n)^T$

3.1.2 基本要素

我们以一个简单的例子来解释线性回归的基本要素。这个应用的目标是预测宝可梦进化后的能力值。我们知道这个能力值取决于很多因素，例如宝可梦进化前的能力值、HP值、MP值的等。为了简单起见，这里我们假设价格只取决于宝可梦进化前的能力，接下来我们探索宝可梦进化后的能力值与进化前的能力值的关系。

模型

设进化前的能力值为x，进化后的能力值为y。我们需要建立基于输入x来计算输出y的表达式，也就是模型（model）。顾名思义，线性回归假设输出与各个输入之间是线性关系：

$$f(x) = xw + b$$

其中w是权重（weight），b是偏差（bias），统称为模型的参数（parameter）。模型输出$f(x)$是线性回归对真实能力值y的预测或估计。通常允许它们之间有一定误差。

损失函数

从上文可知，线性回归试图学得

$$f(x_i) =$$

3.2 线性回归综述-实战

3.2.1 生成数据集

我们构造一个简单的人工训练数据集，它可以使我们能够直观比较学到的参数和真实的模型参数的区别。设数据集样本数为10，输入个数（特征数）为1。给定随机生成的批量样本特征$\boldsymbol{X} \in \mathbb{R}^{10\times 1}$，我们使用线性回归模型真实权重$\boldsymbol{w} = 2$和偏差$b = 4.2$，以及一个随机噪音项$\epsilon$来生成标签

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + b + \epsilon,$$

其中噪音项ϵ服从均值为0和标准差为0.3的正态分布。下面，让我们生成数据集并查看数据分布情况。

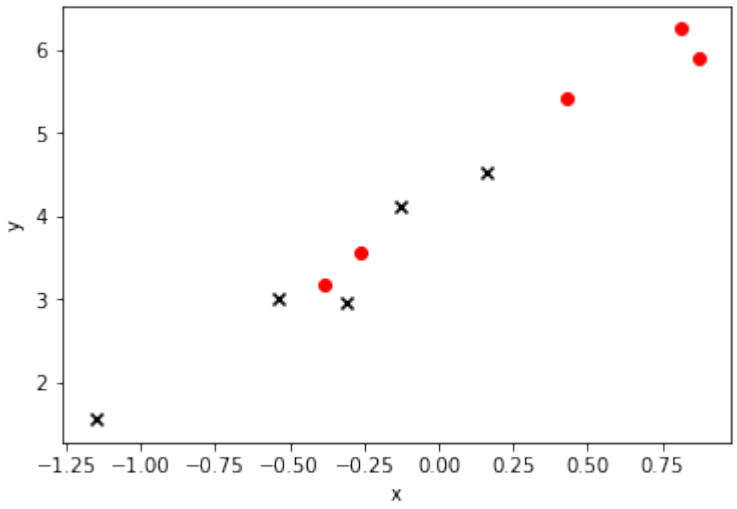
[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

(下页继续)

(续上页)

```
%matplotlib inline

np.random.seed(1)
true_w = 2
true_b = 4.2
x = np.random.normal(scale=0.5, size=(10, 1))
y = true_w * x + true_b
y += np.random.normal(scale=0.3, size=y.shape)

# 使用sklearn.model_selection里的train_test_split模块用于分割数据。
from sklearn.model_selection import train_test_split
# 随机采样50%的数据用于测试，剩下的50%用于构建训练集合。
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.
↪5, random_state=33)

plt.scatter(x_train, y_train, marker='o',c = 'red')
plt.scatter(x_test, y_test, marker='x',c = 'black')
plt.xlabel("x")
plt.ylabel("y")
```

[1]:
```
Text(0, 0.5, 'y')
```



[2]:
```
from sklearn.linear_model import LinearRegression
```

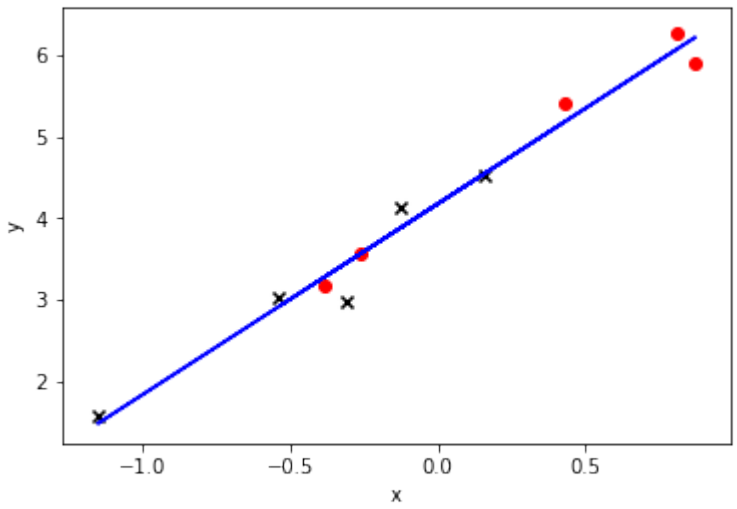
(下页继续)

(续上页)

```
lr = LinearRegression()
lr.fit(x_train,y_train)
y_pred_1 = lr.predict(x_test)
y_pred_0 = lr.predict(x_train)
print('w: %.3f' % lr.coef_[0][0])
print('b: %.3f' % lr.intercept_[0])
e_train = sum((y_pred_0 - y_train)**2) / (2*len(y_train))
print('error(train): %.3f' % ( sum((y_pred_0 - y_train)**2) / (2*len(y_
↪train))))
print('error(test): %.3f' % ( sum((y_pred_1 - y_test)**2) / (2*len(y_
↪test))))

plt.plot(x, lr.coef_[0][0] * x + lr.intercept_[0], 'b-')
plt.scatter(x_train, y_train, marker='o',c = 'red')
plt.scatter(x_test, y_test, marker='x',c = 'black')
plt.xlabel("x")
plt.ylabel("y")
```

```
w: 2.343
b: 4.172
error(train): 0.020
error(test): 0.031
```

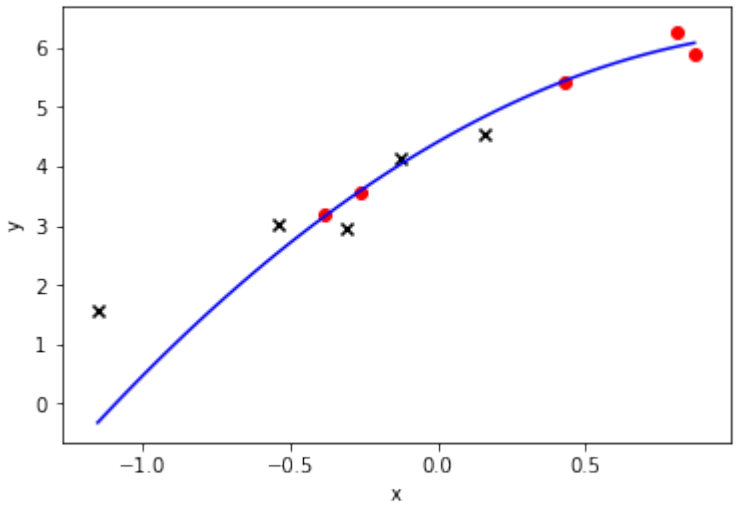
[2]:
```
Text(0, 0.5, 'y')
```



[3]:
```
x1 = np.concatenate((x,x**2), axis=1)

# 使用sklearn.model_selection里的train_test_split模块用于分割数据。
from sklearn.model_selection import train_test_split
# 随机采样50%的数据用于测试，剩下的50%用于构建训练集合。
x_train, x_test, y_train, y_test = train_test_split(x1, y, test_size=0.
↪5, random_state=33)

from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(x_train,y_train)
y_pred_1 = lr.predict(x_test)
y_pred_0 = lr.predict(x_train)
print('w: ', lr.coef_[0])
print('b: ', lr.intercept_[0])
print('error(train): %.3f' % ( sum((y_pred_0 - y_train)**2) / (2*len(y_
↪train))))
print('error(test): %.3f' % ( sum((y_pred_1 - y_test)**2) / (2*len(y_
↪test))))

x1 = np.arange(min(x1[:,0]), max(x1[:,0]), 0.02).reshape(-1, 1)
plt.plot(x1, np.dot(np.concatenate((x1,x1**2), axis=1),lr.coef_[0].
↪reshape(2,1)) + lr.intercept_[0], 'b-')
plt.scatter(x_train[:,0], y_train, marker='o',c = 'red')
plt.scatter(x_test[:,0], y_test, marker='x',c = 'black')
plt.xlabel("x")
plt.ylabel("y")
```

```
w:  [ 2.86658688 -1.08525663]
b:  4.413751215791503
error(train): 0.010
error(test): 0.410
```

[3]:
```
Text(0, 0.5, 'y')
```



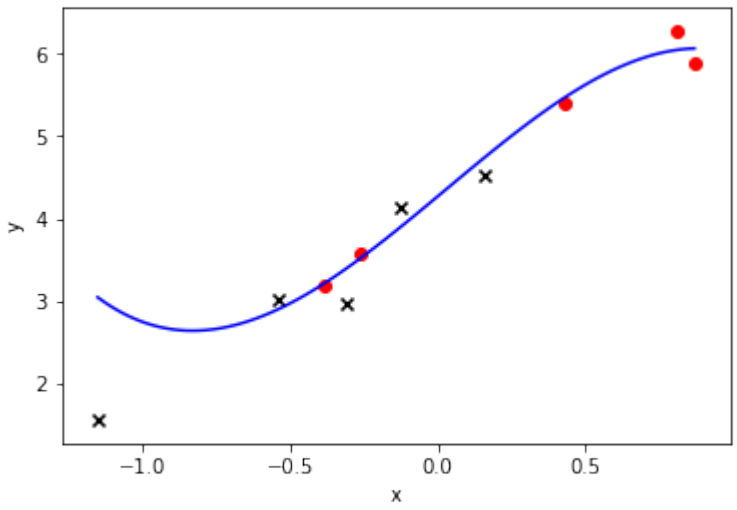
[4]:
```
x1 = np.concatenate((x,x**2,x**3), axis=1)

# 使用sklearn.model_selection里的train_test_split模块用于分割数据。
from sklearn.model_selection import train_test_split
# 随机采样50%的数据用于测试，剩下的50%用于构建训练集合。
x_train, x_test, y_train, y_test = train_test_split(x1, y, test_size=0.
↪5, random_state=33)

from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(x_train,y_train)
y_pred_1 = lr.predict(x_test)
y_pred_0 = lr.predict(x_train)
print('w: ', lr.coef_[0])
print('b: ', lr.intercept_[0])
print('error(train): %.3f' % ( sum((y_pred_0 - y_train)**2) / (2*len(y_
↪train))))
print('error(test): %.3f' % ( sum((y_pred_1 - y_test)**2) / (2*len(y_
↪test))))

x2 = np.arange(min(x1[:,0]), max(x1[:,0]), 0.02).reshape(-1, 1)
plt.plot(x2, np.dot(np.concatenate((x2,x2**2,x2**3), axis=1),lr.coef_
↪[0].reshape(3,1)) + lr.intercept_[0], 'b-')
```

(下页继续)

(续上页)

```
plt.scatter(x_train[:,0], y_train, marker='o',c = 'red')
plt.scatter(x_test[:,0], y_test, marker='x',c = 'black')
plt.xlabel("x")
plt.ylabel("y")
image = plt.show
```

```
w:  [ 2.99088161  0.11409154 -1.35698602]
b:  4.267593723570964
error(train): 0.009
error(test): 0.247
```



3.3 梯度下降法的数学原理

梯度下降法（Gradient descent）是一个一阶最优化算法，通常也称为最速下降法。

要使用梯度下降法找到一个函数的局部极小值，必须向函数上当前点对应梯度（或者是近似梯度）的反方向的规定步长距离点进行迭代搜索。如果相反地向梯度正方向迭代进行搜索，则会接近函数的局部极大值点；这个过程则被称为梯度上升法。

梯度下降方法基于以下的观察：如果实值函数$F(x)$在点a处可微且有定义，那么函数$F(x)$在点a沿着梯度相反的$-\nabla F(a)$下降最快。因此，如果

$$b = a - \lambda \nabla F(a)$$

对于$\lambda > 0$且λ是一个足够小的数值时成立，那么$F(a) \geq F(b)$。

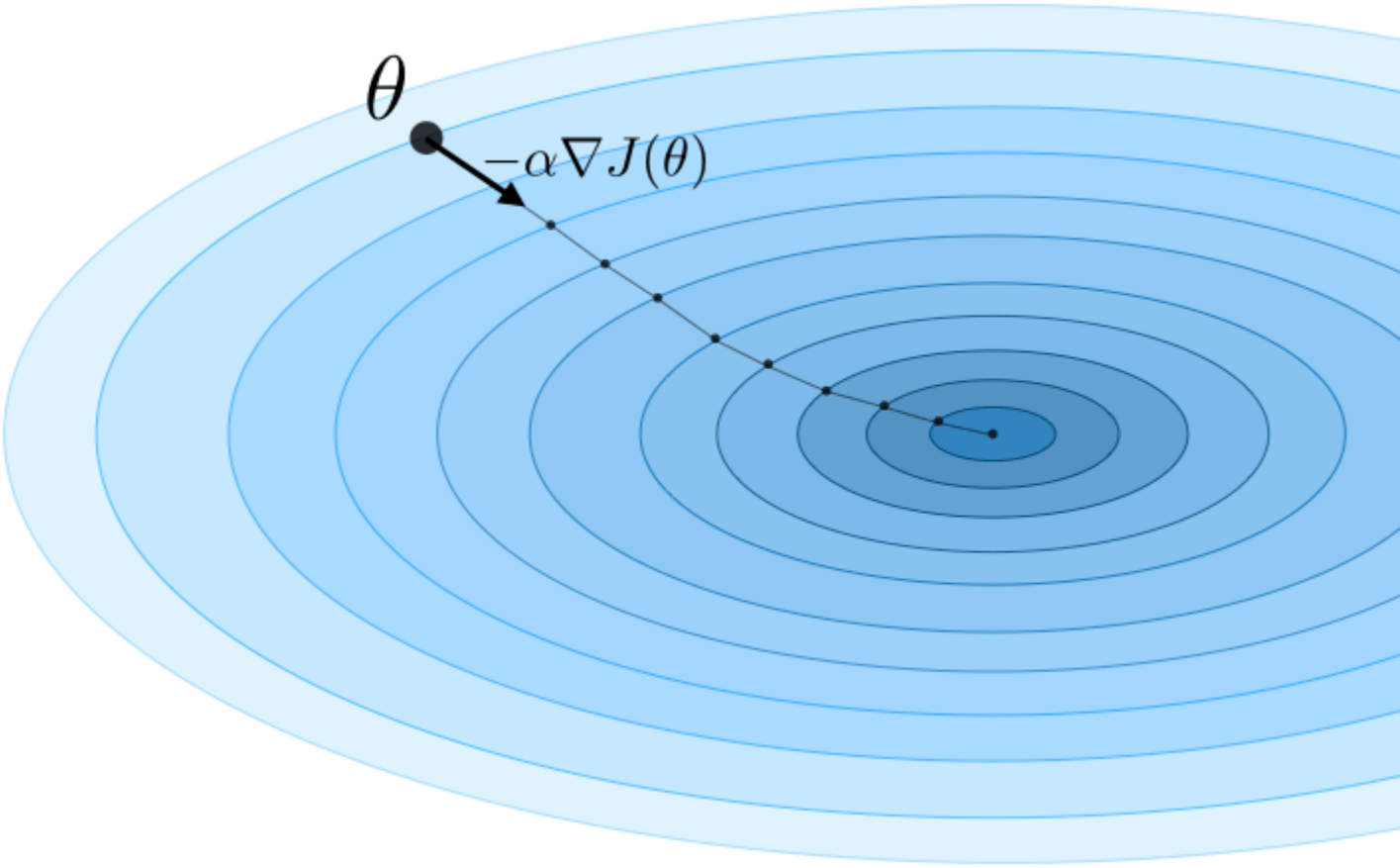
考虑到这一点，我们可以从函数F的初始x_0出发，并考虑如下序列x_0, x_1, x_2...,使得

$$x_{n-1} = x_n - \lambda_n \nabla F(x_n), n \geq 0$$

因此可得到

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \dots$$

如果顺利的话(x_n)就能收敛到期望的极值。其中每次迭代中λ_n的值可以改变。



上图示例了这一过程，这里假设F定义在平面上，并且函数图像是一个碗形。蓝色的曲线是等高线(水平集)，即函数F 为常数的集合构成的曲线。箭头指向该点梯度的反方向。(注：一点处的梯度方向与通过该点的等高线垂直)。沿着梯度下降方向，将最终到达碗底，即函数F值最小的点。

3.3.1 证明

证明：如果实值函数$F(x)$在点a处可微且有定义，那么函数$F(x)$在点a沿着梯度相反的$-\nabla F(a)$下降最快。

提到梯度，就必须从导数、偏导数和方向导数讲起，弄清楚这些概念，才能够正确理解为什么在优化问题中能够使用梯度下降法来优化目标函数。

在这里先简要介绍一下导数和偏导数。

在微积分中，导数反映的是函数$y = f(x)$在某一点处沿x轴正方向的变化率。而偏导数与导数在本质上是一致的，都是当自变量的变化量趋于0时，函数值的变化量与自变量变化量比值的极限。直观地说，偏导数也就是函数在某一点上沿坐标轴正方向的的变化率。

二者区别主要在于：

- 导数，指的是一元函数中，函数$y = f(x)$在某一点处沿x轴正方向的变化率；
- 偏导数,指的是多元函数中,函数$y = f(x_1, x_2, ..., x_n)$在某一点处沿某一坐标轴$(x_1, x_2, ..., x_n)$正方向的变化率。

简要介绍一下导数和偏导数之后，我们主要介绍一下方向导数和梯度，包含完整的推导公式。

现在我们先来讨论函数$z = f(x, y)$在一点P沿某一方向的变化率问题。

为了解决这个问题，我们得引入如下定义：

设函数$z = f(x, y)$在点$P(x, y)$在某一领域 $U(p)$内有定义,从点P引一条射线ll ,设x轴正向到射线l的转角为φ，并设$P'(x + \Delta x, y + \Delta y)$为$l$上的另一点且$P' \in U(p)$ 。我们考虑函数的增量$f(x + \Delta x, y + \Delta y) - f(x, y)$与$)P, P'$两点间距$\rho = \sqrt{(\Delta x)^2 + (\Delta y)^2}$的比值，当$P'$沿着$l$趋于$P$时，如果这个比的极限存在，则称这极限为函数$f(x, y)$在点$P$沿方向$l$的方向导数，记做$\frac{\partial f}{\partial l}$，即：

$$\frac{\partial f}{\partial l} = \lim_{\rho \to 0} \frac{f(x + \Delta x, y + \Delta y) - f(x, y)}{\rho}$$

从定义可知，当函数$f(x, y)$在点$P(x, y)$的偏导数f_x, f_y存在时，函数在点P沿着x轴正向$e_1 = (1, 0)$，y轴正向$e_2 = (0, 1)$的方向导数存在且其值依次为f_x, f_y，函数在点沿x轴负向$e_1^{'} = (-1, 0)$，y轴负向$e_2^{'} = (0, -1)$的方向导数也存在且其值依次为$-f_x, -f_y$。

关于方向导数$\frac{\partial f}{\partial l}$的存在及计算，我们有如下定理：

如果$z = f(x, y)$在点$P(x, y)$是可微的，那么函数在该点沿任一反向的方向导数都存在，且有

$$\frac{\partial f}{\partial l} = \frac{\partial f}{\partial x} cos\varphi + \frac{\partial f}{\partial y} sin\varphi$$

其中φ为x轴到方向l的转角。

证：根据函数$z = f(x, y)$在点$P(x, y)$可微分的假定，函数的增量可以表达为：

$$f(x + \Delta x, y + \Delta y) - f(x, y) = \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial y} \Delta y + o(\rho)$$

两边各除以ρ，得到

$$\begin{aligned}\frac{f(x+\Delta x,y+\Delta y)-f(x,y)}{\rho} &= \frac{\partial f}{\partial x}\frac{\Delta x}{\rho}+\frac{\partial f}{\partial y}\frac{\Delta y}{\rho}+\frac{o(\rho)}{\rho}\\ &= \frac{\partial f}{\partial x}\cos\varphi+\frac{\partial f}{\partial y}\sin\varphi+\frac{o(\rho)}{\rho}\end{aligned}$$

根据

$$\frac{\partial f}{\partial l}=\lim_{\rho\to 0}\frac{f(x+\Delta x,y+\Delta y)-f(x,y)}{\rho}$$

我们就可以证明方向导数存在且其值为

$$\frac{\partial f}{\partial l}=\frac{\partial f}{\partial x}\cos\varphi+\frac{\partial f}{\partial y}\sin\varphi$$

对于三元函数$u=f(x,y,z)$来说，它在空间一点$P(x,y,z)$沿着方向l(设方向的方向角为 (α,β,γ)的方向导数，同样可以定义为

$$\frac{\partial f}{\partial l}=\lim_{\rho\to 0}\frac{f(x+\Delta x,y+\Delta y,z+\Delta z)-f(x,y,z)}{\rho}$$

其中$\rho=\sqrt{(\Delta x)^2+(\Delta y)^2+(\Delta z)^2},\ \Delta x=\rho\,cos\alpha,\Delta y=\rho\,cos\beta,\Delta z=\rho\,cos\gamma$。

同样可以证明，如果函数在所考虑的点处可微分，那么函数在该点沿着l方向的方向导数为:

$$\frac{\partial f}{\partial l}=\frac{\partial f}{\partial x}\cos\alpha+\frac{\partial f}{\partial y}\sin\beta+\frac{\partial f}{\partial z}\cos\gamma$$

同样可以扩展到n元函数$u=f(x_1,x_2,...,x_n)$中，这里就不一一陈诉了。

与方向导数有关联的一个概念是函数的梯度。其定义为:

设函数$z=f(x,y)$在平面区域D内具有一阶连续偏导数，则对于每一点$(x,y)\in D$，都可定义出一个向量$\frac{\partial f}{\partial x}i+\frac{\partial f}{\partial y}j$，这向量称为函数$z=f(x,y)$在点$P(x,y)$的梯度，记作$grad\ f(x,y)$,即

$$grad\ f(x,y)=\frac{\partial f}{\partial x}i+\frac{\partial f}{\partial y}j$$

如果设$e=cos\varphi\,i+sin\varphi\,j$是与方向$l$同方向的单位向量，则由方向导数的计算公式可知:

$$\begin{aligned}\frac{\partial f}{\partial l} &= \frac{\partial f}{\partial x}cos\varphi+\frac{\partial f}{\partial y}sin\varphi\\ &= \left\{\frac{\partial f}{\partial x},\frac{\partial f}{\partial y}\right\}*\{cos\varphi,sin\varphi\}\\ &= grad\ f(x,y)*e\\ &= |grad\ f(x,y)|*cos(grad\ f(x,y),e)\end{aligned}$$

其中$(grad\ f(x,y),e)$表示向量$grad\ f(x,y)$与e的夹角。

由此可以看出，方向导数就是梯度在射线上的投影，当方向l与梯度的方向一致时，有

$$cos\left(grad\, f(x,y), e\right) = 1$$

从而有$\frac{\partial f}{\partial l}$最大值。沿梯度方向的方向导数达到最大值，也就是说梯度的方向是函数$f(x,y)$在这点增长最快的方向。因此，我们可以得到如下结论：函数在某点的梯度是这样一个向量，它的方向与取得最大方向导数的方向一致，而它的模为方向导数的最大值。

接着我们来证明：一点处的梯度方向与通过该点的等高线垂直。

由梯度的定义可知，梯度的模为

$$|grad\, f(x)| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$$

当$\frac{\partial f}{\partial x}$不为零时，那么$x$轴到梯度的转角的正切值为

$$\tan\theta = \frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}$$

我们知道，一般说来二元函数$z = f(x,y)$在几何上表示一个曲面，这曲面被平面$z = c$ (z是常数)所截得的曲线的方程为

$$\begin{cases} z = f(x,y) \\ z = c \end{cases}$$

这条直线l在xOy面上的投影是一条平面曲线l,它在xOy平面直角坐标系中的方程为:

$$f(x,y) = c$$

对于曲线l上的一切点，已给函数的函数值都是c，所以我们称平面曲线l为函数$z = f(x,y)$的等高线。

由于等高线$f(x,y) = c$上任一点(x,y)处的发现的斜率为:

$$-\frac{1}{\frac{\partial y}{\partial x}} = -\frac{1}{\frac{\partial x}{\partial y}} = \frac{f_y}{f_x}$$

所以梯度$\frac{\partial f}{\partial x} * i + \frac{\partial f}{\partial y} * j$为等高线上点$P$处的法向量，因此我们可得到梯度与等高线的下述关系：函数$z = f(x,y)$在点$P(x,y)$的梯度的方向与过点P的等高线$f(x,y) = c$在这点的法线的一个方向相同，且从数值较低的等高线指向数值较高的等高线，而且梯度的模等于函数在这个法线方向的方向导数，这个法线方向就是方向导数取得最大值的方向。

3.4 正则化对线性回归的改进

在我们之前介绍过的优化算法中，目标函数自变量的每一个元素在相同时间步都使用同一个学习率来自我迭代。举个例子，假设目标函数为f，自变量为一个二维向量$[x_1, x_2]^\top$，该向量中每一个元素在迭代时都使用相同的学习率。例如在学习率为η的梯度下降中，元素x_1和x_2都使用相同的学习率η来自我迭代：

在我们之前介绍过的优化算法中，目标函数自变量的每一个元素在相同时间步都使用同一个学习率来自我迭代。举个例子，假设目标函数为f，自变量为一个二维向量$[x_1, x_2]^\top$，该向量中每一个元素在迭代时都使用相同的学习率。例如在学习率为η的梯度下降中，元素x_1和x_2都使用相同的学习率η来自我迭代：

3.5 SGD和自适应学习率对梯度下降法的改进

4

逻辑回归

4.1 逻辑回归综述

4.2 逻辑回归的数学原理

5

感知器

5.1 感知器综述

5.2 感知器的数学原理

6

神经网络

6.1 神经网络

7

支持向量机

7.1 支持向量机综述

7.2 支持向量机的数学原理

8

KNN

8.1 KNN

9

朴素贝叶斯

9.1 朴素贝叶斯

10

决策树

10.1 决策树

11

随机森林

11.1 随机森林

12

K-Means

12.1 K-Means