《人工智能程序设计》

-常用工具库



华为技术有限公司

**目录**

[1 实验介绍 1](#_Toc68178663)

[1.1 实验目的 1](#_Toc68178664)

[1.2 实验清单 1](#_Toc68178665)

[1.3 开发平台介绍 2](#_Toc68178666)

[2 NumPy 2](#_Toc68178667)

[2.1 实验介绍 2](#_Toc68178668)

[2.2 资源准备 3](#_Toc68178669)

[2.3 实验过程 8](#_Toc68178670)

[2.3.1 数组 8](#_Toc68178671)

[2.3.2 矩阵 14](#_Toc68178672)

[2.3.3 随机模块 18](#_Toc68178673)

[2.3.4 常用函数 19](#_Toc68178674)

[2.3.5 向量化编程 21](#_Toc68178675)

[2.4 实验小结 22](#_Toc68178676)

[3 Pandas 22](#_Toc68178677)

[3.1 实验介绍 22](#_Toc68178678)

[3.2 实验过程 23](#_Toc68178679)

[3.2.1 Series 23](#_Toc68178680)

[3.2.2 DataFrame 24](#_Toc68178681)

[3.3 实验小结 33](#_Toc68178682)

[4 Matplotlib 33](#_Toc68178683)

[4.1 实验介绍 33](#_Toc68178684)

[4.2 实验过程 34](#_Toc68178685)

[4.3 实验小结 40](#_Toc68178686)

[5 Scipy 41](#_Toc68178687)

[5.1 实验介绍 41](#_Toc68178688)

[5.2 实验过程 41](#_Toc68178689)

[5.2.1 Scipy 41](#_Toc68178690)

[5.3 ​实验小结 46](#_Toc68178691)

[6 Scikit-learn 47](#_Toc68178692)

[6.1 实验介绍 47](#_Toc68178693)

[6.2 实验过程 47](#_Toc68178694)

[6.2.1 特征工程 47](#_Toc68178695)

[6.2.2 回归算法 48](#_Toc68178696)

[6.2.3 分类算法 53](#_Toc68178697)

[6.2.4 聚类算法 56](#_Toc68178698)

[6.3 实验小结 59](#_Toc68178699)

[7 MindSpore 60](#_Toc68178700)

[7.1 实验介绍 60](#_Toc68178701)

[7.2 资源准备 60](#_Toc68178702)

[7.3 实验过程 61](#_Toc68178703)

[7.3.1 手写数字识别 61](#_Toc68178704)

[7.4 实验总结 75](#_Toc68178705)

# 实验介绍

本实验主要围绕Python常用第三工具库展开实验，包含了Python中用于数据处理、矩阵计算、机器学习和深度学习等方面工具的使用。

## 实验目的

本章实验的主要目的是掌握Python人工智能相关工具库的使用。

## 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验** | **简述** | **难度** | **软件环境** | **开发环境** |
| Numpy | 熟悉numpy工具包的使用，掌握数组、矩阵的运算和常用数学公式的计算。 | 中级 | Python3.7 | ModelArts/ PC 64bit |
| Pandas | 掌握pandas中常用数据结构的使用，掌握pandas中常用的数据处理函数。。 | 中级 | Python3.7 | ModelArts/ PC 64bit |
| Matplotlib | 使用matplotlib完成图形的绘制。 | 中级 | Python3.7 | ModelArts/ PC 64bit |
| Scipy | 使用scipy完成常用数学函数的计算。 | 中级 | Python3.7 | ModelArts/ PC 64bit |
| scikit-learn | 掌握机器学习工具scikit-learn的基本使用。 | 中级 | Python3.7- | ModelArts/ PC 64bit |
| MindSpore | 了解华为自研快加MindSpore，掌握使用MindSpore构建模型并训练 | 中级 | ModelArts  MindSpore | ModelArts/ PC 64bit |

## 开发平台介绍

参考课程ppt胶片。

# NumPy

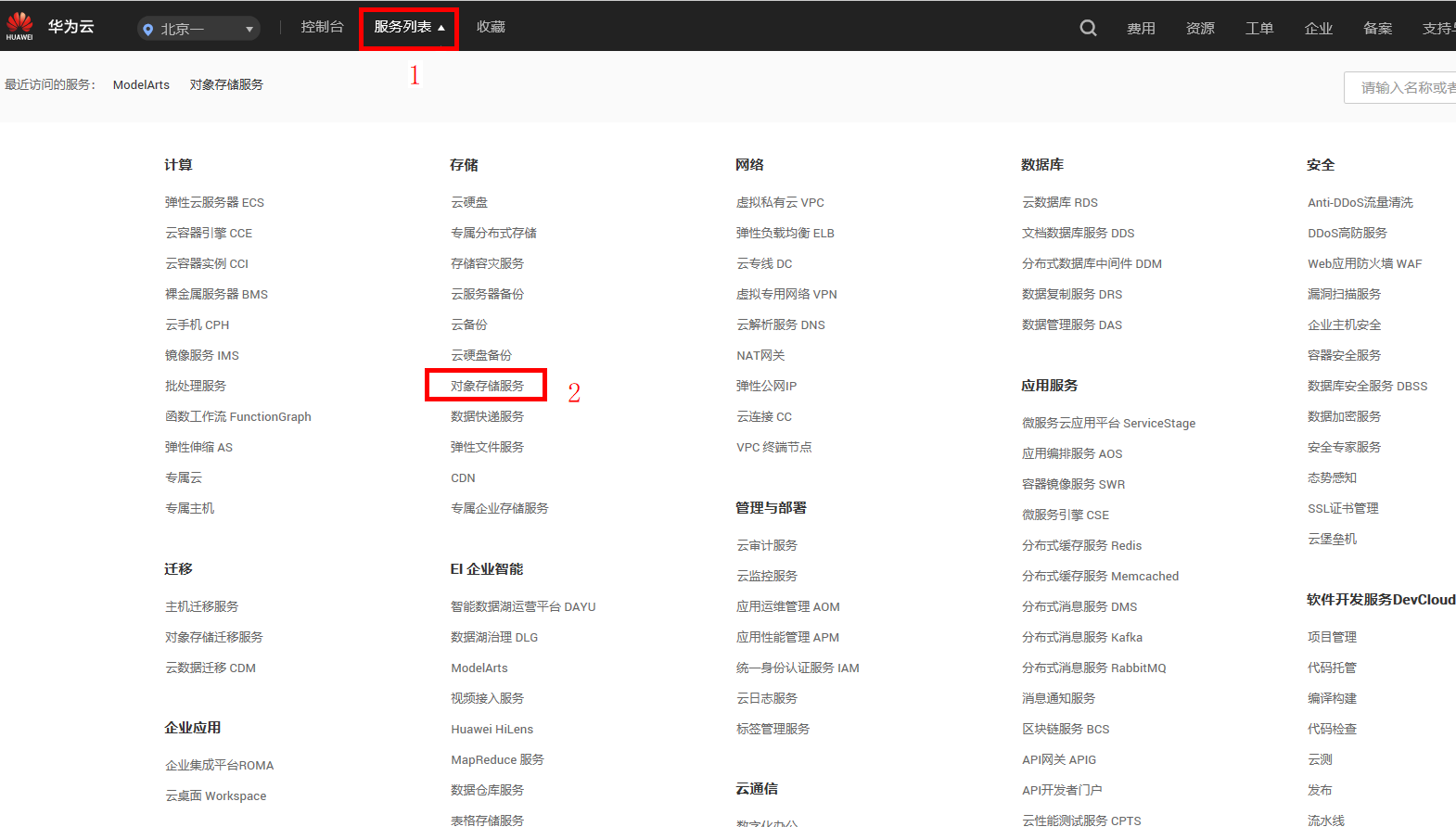
## 实验介绍

本实验帮助大家了解Python中用于科学计算的工具 - Numpy。包括了数组、矩阵、随机模块和一些常用的数学公式。

## 资源准备

资源准备

登陆华为云[https://auth.huaweicloud.com/login.action#/login](https://auth.huaweicloud.com/login.action" \l "/login)后，点击右上角的“控制台”，找到“服务列表”下的“对象存储服务”：



对象存储服务位置

点击“对象存储服务”按钮，进入对象存储服务的界面，右上角点击“创建桶”，这是为了存储数据。



对象存储服务界面

按默认方式点击右下角“立即创建”：



创建桶的界面

创建好之后，见下图：



创建桶之后

点击新创建的桶，找到“对象”后点击“新建文件夹”，命名为movie\_review：



新建文件夹

进入movie\_review文件夹，然后新建两个文件夹，model和data，用于存放模型和数据。



上传文件

上传数据：



上传文件

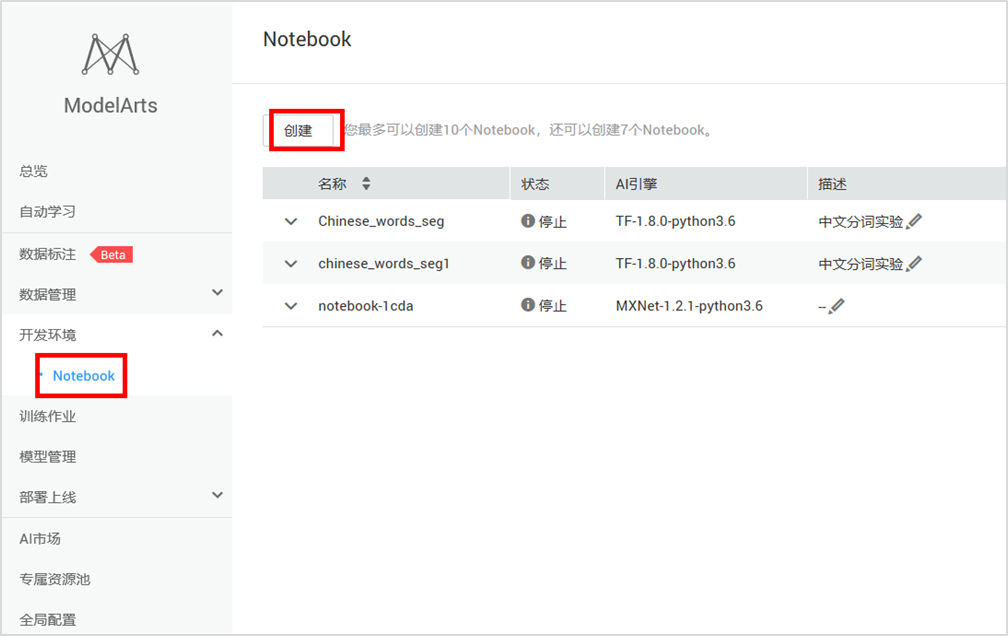
ModelArts环境准备

从服务列表中找到ModelArts，也可以从服务列表中进行搜索：



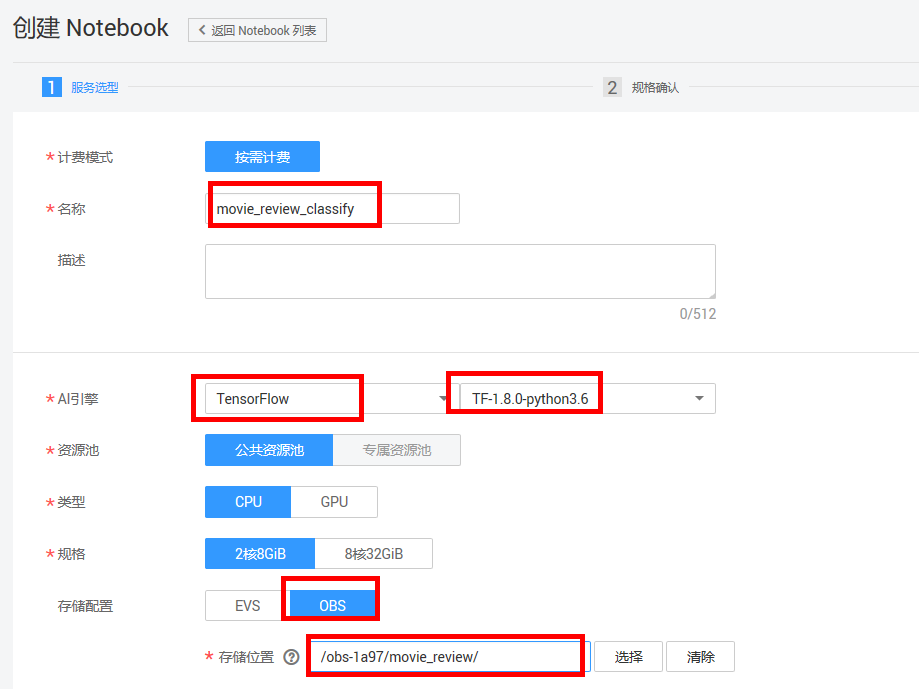
搜索ModelArts

从“开发环境”中创建“Notebook”，以下图为例：



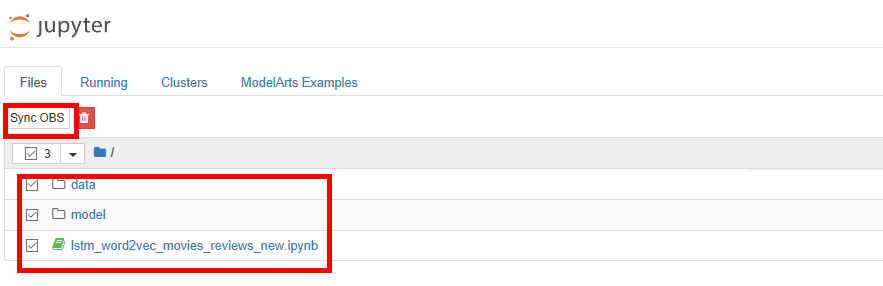
创建Notebook

为Notebook配置所需要的资源，并配置存储位置：



配置Notebook的信息

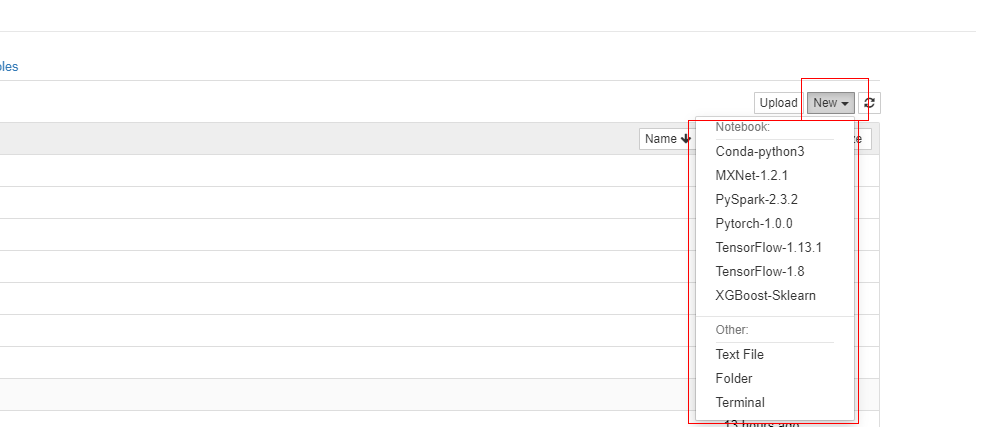
启动该Notebook后，新建ipynb文件然后，选中下面3个文件后点击”Sync OBS”进行同步数据。



同步数据

然后就可以在ipynb中进行访问数据和模型了。

创建ipynb进行代码编写。点击New，选择使用的解释器。



创建ipynb文件

本实验中根据实验的不同选择的 也不相同。（实验1-实验6，可以选择conda-Python3或者Tensorflow，实验7,8选择Tensorflow1.13.1，实验9选择pytorch1.0.0，如果出现导包的问题，可以尝试更换一下解释器）

## 实验过程

### 数组

导入numpy模块

代码如下：

import numpy as np

步骤 2 创建数组

|  |  |
| --- | --- |
| **数组生成方法（numpy简写为np）** | **使用详情** |
| np.array(object, dtype=None, copy=True,) | 创建一个数组对象。 |
| np.zeros(shape, dtype=float, order='C') | 创建一个形状为shape的全零数组。dtype为数据类型。order=C代表与c语言类似，行优先;order=F代表列优先 |
| np.ones(shape, dtype=None, order='C') | 创建一个全1数组，和np.zeros()类似。 |
| np.eye(N, M=None, k=0, dtype=float, order='C') | 生成一个类似于对角矩阵的数组，N为行数；M为列数，默认和N一样；k为对角线的索引，0代表主对角线。 |
| np.empty(shape, dtype=float, order='C') | 生成一个未初始化的数组 |
| np.asarray(a, dtype = None, order = None) | 将数据转化为数组。a:输入数据,dtype：数据类型，一般默认为数据本身类型，可选（区别于array，asarray会直接修改源数据） |
| np.arange(start, stop, step, dtype) | 生成一个等间隔的数组。  start：起始范围，默认为0，；stop：终止范围  step：前后间隔，默认为1；dtype：数据类型。 |
| np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0) | 生成一个等间隔的数组，start起始值，stop终止值，num数量，endpoint=True表示stop为最后一个值。 |

示例：

a = np.array([1,2,3,4,5,6]) # 一维数组

print("一维数组：", a)

b = np.array([[1,2,3],[4,5,6]]) # 多维数组

print("多维数组：" ,b)

c = np.zeros([2,2]) #创建一个2\*2的全0数组

print("全零数组：",c)

d = np.ones([2,2]) # 创建一个2\*2的全1数组

print("全一数组：",d)

print(np.eye(2))

e = np.asarray([1,2,3,4,5,6]) # 一维数组

print("一维数组：", e)

f = np.arange(10,20,2) # 一维等间隔数组

print("等间隔数组")

g = np.linspace(0,100,10) #一维等间隔数组

print(g)

输出结果：

一维数组： [1 2 3 4 5 6]

多维数组： [[1 2 3]

[4 5 6]]

全零数组： [[0. 0.]

[0. 0.]]

全一数组： [[1. 1.]

[1. 1.]]

[[1. 0.]

[0. 1.]]

一维数组： [1 2 3 4 5 6]

等间隔数组

[ 0. 11.11111111 22.22222222 33.33333333 44.44444444

55.55555556 66.66666667 77.77777778 88.88888889 100. ]

数组属性

a = np.array([[1,2],[3,4],[5,6]])

print('shape =',a.shape) #数组的维度

print('shape\_row =',a.shape[0]) #数组第一维度上的大小,即行数

print('size =',a.size) #数组元素的个数

print('size\_column =',np.size(a,0)) #行数

print('len =',len(a)) #数组第一维度的大小，即行数

print(a.ndim) # 数组维度·

b = a.reshape(2,3)

print(b.ndim)

输出结果：

shape = (3, 2)

shape\_row = 3

size = 6

size\_column = 3

len = 3

1

2

数组常用操作

a = np.arange(10) # 创建数组

print(a[2:8:2]) # 切片 下标为2开始到8结束（不取8），步长为2

print(a[2:8])

print(a[2])

print(a[2:])

输出结果：

[2 4 6]

[2 3 4 5 6 7]

2

[2 3 4 5 6 7 8 9]

布尔索引：根据True或False选择原数组中的数据。

a = np.array([[1, 2], [3, 4]])

print(a>2)

print (a[a > 2])

输出结果：

[[False False]

[ True True]]

[3 4]

reshape操作：改变数组维度。

a = np.array([[1,2],[3,4],[5,6]])

print("a:\n",a)

b = a.reshape(2,3)

print("reshape:\n",b)

输出结果：

a:

[[1 2]

[3 4]

[5 6]]

reshape:

[[1 2 3]

[4 5 6]]

输出结果：

广播

广播(Boardcasting)是NumPy中用于在不同大小的阵列（包括标量与向量，标量与二维数组，向量与二维数组，二维数组与高维数组等）之间进行逐元素运算（例如，逐元素 加法，减法，乘法，赋值等）的一组规则。

两个array的shape长度与shape的每个对应值都相等的时候，那么结果就是对应元素逐元素运算，运算的结果shape不变。

a = np.array([1,2,3,4])

b = np.array([1,2,3,4])

c = a \* b

print(c)

输出结果：

[ 1 4 9 16]

shape长度不相等时，先把短的shape前面一直补1，直到与长的shape长度相等时，此时，两个array的shape对应位置上的值 ：1、相等 或 2、其中一个为1，这样才能进行广播。

a = np.arange(27).reshape((3,3,3))

print("a:\n",a)

b = np.array([[1,2,3]])

c = a + b

print("广播之后：\n",c)

输出结果：

a:

[[[ 0 1 2]

[ 3 4 5]

[ 6 7 8]]

[[ 9 10 11]

[12 13 14]

[15 16 17]]

[[18 19 20]

[21 22 23]

[24 25 26]]]

广播之后：

[[[ 1 3 5]

[ 4 6 8]

[ 7 9 11]]

[[10 12 14]

[13 15 17]

[16 18 20]]

[[19 21 23]

[22 24 26]

[25 27 29]]]

常用函数

flat函数：按照下标取元素，在多维数组中也可以直接取到对应的元素。

a = np.arange(6).reshape(2,3)

print("a\n",a)

print("a[1]\n",a[1])

print("a.flat[1]\n",a.flat[1])

输出结果：

a

[[0 1 2]

[3 4 5]]

a[1]

[3 4 5]

a.flat[1]

1

transpose函数：根据数组的行列索引值对数据进行转换（二维数组默认是转置）。

a = np.arange(12).reshape(2,2,3)

b=np.transpose(a,(2,0,1))

print('a=',a)

print('b=',b)

输出结果：

a= [[[ 0 1 2]

[ 3 4 5]]

[[ 6 7 8]

[ 9 10 11]]]

b= [[[ 0 3]

[ 6 9]]

[[ 1 4]

[ 7 10]]

[[ 2 5]

[ 8 11]]]

concentrate（np.concatenate((a1,a2,a3,...), axis=0)）函数：按照特定方向轴进行拼接，默认是列：

a = np.arange(6).reshape(2,3)

b = np.arange(7,13).reshape(2,3)

c = np.concatenate((a,b))

d = np.concatenate((a,b),axis = 1) # axis=1：按行拼接

print('a=',a)

print('b=',b)

print('c=',c)

print('d=',d)

输出结果：

a= [[0 1 2]

[3 4 5]]

b= [[ 7 8 9]

[10 11 12]]

c= [[ 0 1 2]

[ 3 4 5]

[ 7 8 9]

[10 11 12]]

d= [[ 0 1 2 7 8 9]

[ 3 4 5 10 11 12]]

stack函数（numpy.stack(arrays,axis=0)）：用于堆叠数组，arrays是用于堆叠的数组，asix堆叠的方向。

a = np.array([[1,2],[3,4]])

b = np.array([[5,6],[7,8]])

c = np.stack((a,b),0)

d = np.stack((a,b),1)

print('a=',a)

print('b=',b)

print('c=',c)

print('d=',d)

输出结果：

a= [[1 2]

[3 4]]

b= [[5 6]

[7 8]]

c= [[[1 2]

[3 4]]

[[5 6]

[7 8]]]

d= [[[1 2]

[5 6]]

[[3 4]

[7 8]]]

delete函数（numpy.delete(arr, obj, axis=None)）：返回删除obj处数据后的值，不会改变原来的数据。

a = np.arange(6).reshape(2,3)

b = np.delete(a,2)

c = np.delete(a,1,axis=1)

print('a=',a)

print('b=',b)

print('c=',c)

输出结果：

a= [[0 1 2]

[3 4 5]]

b= [0 1 3 4 5]

c= [[0 2]

[3 5]]

### 矩阵

步骤 1 导入numpy模块

import numpy.matlib

import numpy as np

创建矩阵

a=np.matlib.zeros((2,2)) # 全一矩阵

b=np.matlib.ones((2,2)) # 全零矩阵

print('a=',a)

print('b=',b)

输出结果:

a= [[0. 0.]

[0. 0.]]

b= [[1. 1.]

[1. 1.]]

对角矩阵：

c=np.matlib.eye(3)

print('c=',c)

输出结果：

c= [[ 1. 0. 0.]

[ 0. 1. 0.]

[ 0. 0. 1.]]

数组转化为矩阵：

d = np.array([[1,2],[3,4]])

d = np.mat(a)

print(d)

print(type(d))

输出结果：

[[1 2]

[3 4]]

<class 'numpy.matrixlib.defmatrix.matrix'>

随机生成矩阵：

a=np.matlib.rand(3,3)

print('a=',a)

输出结果：

a= [[ 0.08480457 0.14825149 0.62123201]

[ 0.98311546 0.35140819 0.42553922]

[ 0.09744874 0.92214506 0.64185678]]

numpy命名空间 mat/matrix/asmatrix函数：

a=np.array([1,2,3])

b=np.mat(a) #三个函数功能类似

d=np.asmatrix(a) #matrix和asmatrix的区别在于asmatrix处理矩阵或数组时不复制

c=np.matrix(a) #类似array和asarray

print('array a=',a)

print('matrix b=',b)

print('matrix c=',c)

print('matrix d=',d)

输出结果：

array a= [1 2 3]

matrix b= [[1 2 3]]

matrix c= [[1 2 3]]

matrix d= [[1 2 3]]

矩阵运算

a=np.mat([[1],[2],[3]])

b=np.mat([1,2,3])

c=a\*b #星乘(\*)和点乘(dot)都是从ndarray类中继承而来的

d=np.dot(a,b) #星乘(\*)对于数组和矩阵而言进行的操作不同

print('matrix a=',a)

print('matrix b=',b)

print('matrix c=',c)

print('matrix d=',d)

输出结果：

matrix a= [[1]

[2]

[3]]

matrix b= [[1 2 3]]

matrix c= [[1 2 3]

[2 4 6]

[3 6 9]]

matrix d= [[1 2 3]

[2 4 6]

[3 6 9]]

对应元素相乘：

a=np.mat([1,2,3])

b=a

c=np.multiply(a,b)

d=a\*2

print('matrix a=',a)

print('matrix b=',b)

print('matrix c=',c)

print('matrix d=',d)

输出结果：

matrix a= [[1 2 3]]

matrix b= [[1 2 3]]

matrix c= [[1 4 9]]

matrix d= [[2 4 6]]

矩阵求逆：

a=np.matrix([[2,0,0],

[0,1,0],

[0,0,2]])

b=a.I

print('matrix a=',a)

print('matrix b=',b)

输出结果：

matrix a= [[2 0 0]

[0 1 0]

[0 0 2]]

matrix b= [[ 0.5 0. 0. ]

[ 0. 1. 0. ]

[ 0. 0. 0.5]]

行列式计算：

a=np.matrix([[1,2],

[3,4]])

b=np.linalg.det(a)

print('matrix a=',a)

print(' b=',b)

输出结果：

matrix a= [[1 2]

[3 4]]

b= -2.0

求和与最大最小：

a=np.matrix([[1,2],

[3,4]])

b=a.sum(axis=0) #对列求和

c=a.sum(axis=1) #对行求和

d=a.max()

e=a.min()

print('matrix a=',a)

print(' b=',b)

print(' c=',c)

print(' d=',d)

print(' e=',e)

输出结果：

matrix a= [[1 2]

[3 4]]

b= [[4 6]]

c= [[3]

[7]]

d= 4

e= 1

矩阵转数组

a=np.mat([[1,2,3],[4,5,6]])

b=a.getA() #getA将矩阵类转化为数组类

print('matrix a=',a)

print(type(a))

print('array b=',b)

print(type(b))

输出结果：

matrix a= [[1 2 3]

[4 5 6]]

<class 'numpy.matrixlib.defmatrix.matrix'>

array b= [[1 2 3]

[4 5 6]]

<class 'numpy.ndarray'>

### 随机模块

导入numpy模块

import numpy as np

简单随机数据

r1=np.random.rand(2,2) #rand函数 范围在[0,1)

r2=np.random.randn(2,2) #randn函数 具有标准正态分布

r3=np.random.randint(0,5) #randint(low,high,size)函数 返回在[low,high)范围的整数

r4=np.random.random((2,2)) #random函数，和random.rand函数输出相同，输入有区别

print('r1=','\n',r1)

print('r2=','\n',r2)

print('r3=','\n',r3)

print('r4=','\n',r4)

输出结果：

r1=

[[ 0.01574989 0.23790136]

[ 0.90032663 0.28935008]]

r2=

[[-0.58652778 1.95238333]

[ 1.5595297 -0.94242376]]

r3=

3

r4=

[[ 0.59990204 0.07215791]

[ 0.56173751 0.19867578]]

r5=

[[ 0.01698654 0.89418447]

[ 0.591243 0.97378358]]

choice函数（choice(a[, size, replace, p])）：

r1=np.random.choice(5, 3) #从np.arange(5)数组中抽取3个数

r2=np.random.choice(5, 3, p=[0.1, 0, 0.3, 0.6, 0]) #被抽取数组中的元素具有不同概率

r3=np.random.choice(5, 3, replace=False) #不放回抽样

print('r1=','\n',r1)

print('r2=','\n',r2)

print('r3=','\n',r3)

输出结果：

r1=

[2 4 2]

r2=

[3 3 3]

r3=

[0 2 4]

随机排列

sample=np.arange(5)

np.random.shuffle(sample) #改变自身内容

r2=np.random.permutation([1,2,3,4]) #返回随机序列

print('sample=','\n',sample)

print('r2=','\n',r2)

输出结果：

sample=

[1 2 4 3 0]

r2=

[3 1 2 4]

常用分布

r1=np.random.normal(0,0.1,5) #正态分布，参数1为均值，参数2为标准差，参数3为返回值的维度

r2=np.random.uniform(0,5,2) #均匀分布

r3=np.random.poisson(5,2) #泊松分布

print('r1=','\n',r1)

print('r2=','\n',r2)

print('r3=','\n',r3)

输出结果：

r1=

[-0.03039277 0.05921658 0.00315102 0.11307963 0.00920257]

r2=

[ 1.77185696 3.45519433]

r3=

[7 3]

### 常用函数

三角函数

a=np.array([0,30,45,60,90])

b=np.sin(a\*np.pi/180)

c=np.sin(a\*np.pi/180)

print('b=',b)

print('c=',c)

输出结果：

b= [ 0. 0.5 0.70710678 0.8660254 1. ]

c= [ 0. 0.5 0.70710678 0.8660254 1. ]

around函数-四舍五入：

a=np.array([1.0,1.5,2.0,2.55])

b=np.around(a)

c=np.around(a,decimals=1)

print('b=',b)

print('c=',c)

输出结果：

b= [ 1. 2. 2. 3.]

c= [ 1. 1.5 2. 2.6]

floor/ceil函数：

a=np.array([1.0,1.5,2.0,2.55])

b=np.floor(a) # 向下取整

c=np.ceil(a) # 向上取整

print('b=',b)

print('c=',c)

输出结果：

b= [ 1. 1. 2. 2.]

c= [ 1. 2. 2. 3.]

算数运算

a=np.array([1,2,3,4])

b=np.array([4,3,2,1])

c=np.add(a,b) #加

d=np.subtract(a,b) #减

e=np.multiply(a,b) #乘

f=np.divide(a,b) #除

g=np.mod(a,b) #取余

h=np.power(a,b) #乘方

print('c=',c)

print('d=',d)

print('e=',e)

print('f=',f)

print('g=',g)

print('h=',h)

输出结果：

c= [5 5 5 5]

d= [-3 -1 1 3]

e= [4 6 6 4]

f= [ 0.25 0.66666667 1.5 4. ]

g= [1 2 1 0]

h= [1 8 9 4]

统计函数

a=np.arange(6).reshape(2,3)

b=np.amin(a,0) #第0维度上最小值

c=np.amax(a,1) #第1维度上最大值

d=np.median(a) #中位数

e=np.mean(a) #平均数

print('a=',a)

print('b=',b)

print('c=',c)

print('d=',d)

print('e=',e)

输出结果：

a= [[0 1 2]

[3 4 5]]

b= [0 1 2]

c= [2 5]

d= 2.5

e= 2.5

排序函数

numpy.sort(a, axis, kind, order)：对数组进行排序，a：要排序的数组，axis：排序轴

a=np.array([[3,5,1],[2,8,7]])

b=np.sort(a)

b=np.sort(a,axis=0)

print(b)

输出结果：

[[2 5 1]

[3 8 7]]

### 向量化编程

某些情况下，在Python中使用for进行循环控制时，效率会很低，为了解决这个问题可以使用向量化编程。

import numpy as np

import time

# 计算两组数据

a = np.random.rand(1000000)

b = np.random.rand(1000000)

# 使用向量化

tic = time.time()

c = np.dot(a, b)

toc = time.time()

print("c: %f" % c)

print("耗时:" + str(1000\*(toc-tic)) + "ms")

输出：

c: 249630.494367

耗时:1.001119613647461ms

使用for循环：

c = 0

tic = time.time()

for i in range(1000000):

c += a[i] \* b[i]

toc = time.time()

print("c: %f" % c)

print("耗时:" + str(1000\*(toc-tic)) + "ms")

输出：

c: 249630.494367

耗时:473.585844039917ms

## 实验小结

本实验主要介绍了 numpy中的两种数据结构数组和矩阵的使用，还有numpy中常用的一些函数和随机模块的使用。

# Pandas

## 实验介绍

在Python中对数据进行处理时，经常会使用到Pandas这个库，它提供了诸多的数据处理方法和时间序列的操作方法，是当下最重要的Python 科学运算模块之一。。

## 实验过程

### Series

创建Series

1) Series可以通过pandas.Series()创建：pandas.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)：生成一个Series数据。data为数据可以是数组和字典等；index为索引值，要求与数据长度相同，dtype为数据类型。

import pandas as pd

s = pd.Series([1,3,5,np.nan,6,8]) # nan 缺失值

输出结果：

0 1.0

1 3.0

2 5.0

3 NaN

4 6.0

5 8.0

dtype: float64

2) 从ndarray创建一个Series。

data = np.array(['a','b','c','d'])

s = pd.Series(data)

print(s)

输出结果：

0 a

1 b

2 c

3 d

dtype: object

3）从字典创建一个Series。

‘’’字典（dict）可以作为输入传递，如果没有指定索引，则按排序顺序取得字典键以构造索引。 如果传递了索引，索引中与标签对应的数据中的值将被取出。’’’

data1 = {'a' : 0., 'b' : 1., 'c' : 2.}

s = pd.Series(data1)

print(s)

输出结果：

a 0.0

b 1.0

c 2.0

dtype: float64

### DataFrame

创建DataFrame

DataFrame既有行索引，也有列索引，可以看做是Series组成的字典，每个Series看做DataFrame的一个列。

1）pandas. DataFrame ()创建。

时间序列生成：pandas.date\_range（start = None，end = None，periods = None，freq = "D"，tz = None，normalize = False，name = None，closed = None，\*\*kwargs ）：生成一个时间序列的索引DatetimeIndex。start为日期起点，end为日期终点，periods为个数，freq表示间隔（D表示以日为间隔）,tz表示时区。

dates = pd.date\_range('20200101', periods=7)# 生成作为行索引的时间序列

print(dates)

print("--"\*16)

# 使用随机生成的numpy数组作为数据，传入列索引ABCD

df = pd.DataFrame(np.random.randn(7,4), index=dates, columns=list('ABCD'))

print(df)

输出结果：

DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',

'2020-01-05', '2020-01-06', '2020-01-07'],

dtype='datetime64[ns]', freq='D')

--------------------------------

A B C D

2020-01-01 0.460328 0.595538 0.961140 0.442340

2020-01-02 -0.695792 -0.982458 -1.219558 -1.652008

2020-01-03 -0.013150 0.801074 -0.425132 2.434673

2020-01-04 1.299650 2.003954 -1.446891 -0.522698

2020-01-05 0.607561 1.083350 -0.780739 -0.524102

2020-01-06 1.080098 0.306991 0.219004 0.553709

2020-01-07 -0.426692 -0.767197 1.025897 -0.573590

2）使用字典创建DataFrame。

df1 = pd.DataFrame({ 'A' : 1.,

'B' : pd.Timestamp('20200102'), # Timestamp 方法生成时间戳

'C' : pd.Series(1,index=list(range(4)),dtype='float32'),

'D' : np.array([3] \* 4,dtype='int32'),

# Categoricals 是 pandas 的一种数据类型，对应着被统计的变量，

# categorical 类型的数据可以具有特定的顺序，这个顺序是创建时手工设定的，是静态的

'E' : pd.Categorical(["test","train","test","train"]),

'F' : 'foo' })

print(df1)

输出结果：

A B C D E F

0 1.0 2020-01-02 1.0 3 test foo

1 1.0 2020-01-02 1.0 3 train foo

2 1.0 2020-01-02 1.0 3 test foo

3 1.0 2020-01-02 1.0 3 train foo

查看DataFrame中的数据

1) 查看顶部和底部的数据。

data2 = np.arange(30).reshape(6,5) # 生成一个维度（6,5）的数组

df2 = pd.DataFrame(data2,index=['a','b','c','d' ,'e' , 'f'] , columns = ['A', 'B', 'C','D','E']) # 创建DataFrame

print(df2)

print("--" \* 10)

# DataFrame.head(n=5)：显示前n条数据。n表示显示的数据量。

print(df2.head())

print("--" \* 10)

# DataFrame.tail(n=5)：显示底部数据，和head的参数一样

print(df2.tail(3))

输出结果：

A B C D E

a 0 1 2 3 4

b 5 6 7 8 9

c 10 11 12 13 14

d 15 16 17 18 19

e 20 21 22 23 24

f 25 26 27 28 29

--------------------

A B C D E

a 0 1 2 3 4

b 5 6 7 8 9

c 10 11 12 13 14

d 15 16 17 18 19

e 20 21 22 23 24

--------------------

A B C D E

d 15 16 17 18 19

e 20 21 22 23 24

f 25 26 27 28 29

2) 显示索引，列和数据。

print("index is :" )

print(df2.index) # 输出行索引

print("columns is :" )

print(df2.columns) # 输出列索引

print("values is :" )

print(df2.values) # 输出数据

输出结果：

index is :

Index(['a', 'b', 'c', 'd', 'e', 'f'], dtype='object')

columns is :

Index(['A', 'B', 'C', 'D', 'E'], dtype='object')

values is :

[[ 0 1 2 3 4]

[ 5 6 7 8 9]

[10 11 12 13 14]

[15 16 17 18 19]

[20 21 22 23 24]

[25 26 27 28 29]]

3) 使用loc方法，根据列名对相关索引进行切片，得到想要访问的数据。

# DataFrame.loc：按标签或布尔数组访问一组行和列

print(df2.loc['a':'f':2,'A']) # 获取A列，索引为a到f（包括f）中的数据，步长为2

输出结果：

a 0

c 10

e 20

Name: A, dtype: int32

4) 查看数据的详细信息。

DataFrame.describe(percentiles=None, include=None, exclude=None)：生成描述性统计数据，总结数据集分布的集中趋势，分散和形状，不包括 NaN值。

print(df2.describe()) # describe描述了数据的详细信息

输出结果：

A B C D E

count 6.000000 6.000000 6.000000 6.000000 6.000000

mean 12.500000 13.500000 14.500000 15.500000 16.500000

std 9.354143 9.354143 9.354143 9.354143 9.354143

min 0.000000 1.000000 2.000000 3.000000 4.000000

25% 6.250000 7.250000 8.250000 9.250000 10.250000

50% 12.500000 13.500000 14.500000 15.500000 16.500000

75% 18.750000 19.750000 20.750000 21.750000 22.750000

max 25.000000 26.000000 27.000000 28.000000 29.000000

# （count：一列的元素个数；

mean：一列数据的平均值；

std：一列数据的均方差；（方差的算术平方根，反映一个数据集的离散程度：越大，数据间的差异越大，数据集中数据的离散程度越高；越小，数据间的大小差异越小，数据集中的数据离散程度越低）

min：一列数据中的最小值；

max：一列数中的最大值；

25%：一列数据中，前 25% 的数据的平均值；

50%：一列数据中，前 50% 的数据的平均值；

75%：一列数据中，前 75% 的数据的平均值；）

DataFrame数据的操作

1） 使用drop方法删除不需要的列或行。

data3 = np.arange(30).reshape(6,5)

df3 = pd.DataFrame(data3,index=['a','b','c','d','e', 'f'] , columns = ['A', 'B', 'C', 'D', 'E'])

a = df3.drop(['a'], axis=0) # axis=0时 删除指定的行

b = df3.drop(['A'], axis=1) # axis=1时，删除指定的列

print('-------原始数据df-----')

print(df3)

print('-------删除行---------')

print(a)

print('-------删除列---------')

print(b)

输出结果：

-------原始数据df---------

A B C D E

a 0 1 2 3 4

b 5 6 7 8 9

c 10 11 12 13 14

d 15 16 17 18 19

e 20 21 22 23 24

f 25 26 27 28 29

-------删除行---------

A B C D E

b 5 6 7 8 9

c 10 11 12 13 14

d 15 16 17 18 19

e 20 21 22 23 24

f 25 26 27 28 29

-------删除列---------

B C D E

a 1 2 3 4

b 6 7 8 9

c 11 12 13 14

d 16 17 18 19

e 21 22 23 24

f 26 27 28 29

2） 使用append方法合并两个DataFrame。

c = b.append(a)

print(b)

print('------合并后产生的新数据------')

print(c)

输出结果：

B C D E

a 1 2 3 4

b 6 7 8 9

c 11 12 13 14

d 16 17 18 19

e 21 22 23 24

f 26 27 28 29

------合并后产生的新数据------

A B C D E

a NaN 1 2 3 4

b NaN 6 7 8 9

c NaN 11 12 13 14

d NaN 16 17 18 19

e NaN 21 22 23 24

f NaN 26 27 28 29

b 5.0 6 7 8 9

c 10.0 11 12 13 14

d 15.0 16 17 18 19

e 20.0 21 22 23 24

f 25.0 26 27 28 29

3）使用reset\_index方法还原索引，让索引变为数据中的一列。

DataFrame.reset\_index（level = None，drop = False，inplace = False，col\_level = 0，col\_fill ='' ）：重置索引。level默认为None仅从索引中删除给定的级别，默认情况下删除所有级别。drop表示是否将索引添加至数据成为一列。

b.reset\_index(inplace=True)# inplace为true时会修改原始数据，为False会产生新的数据

print(b)

输出结果：

index B C D E

0 a 1 2 3 4

1 b 6 7 8 9

2 c 11 12 13 14

3 d 16 17 18 19

4 e 21 22 23 24

5 f 26 27 28 29

iteritems方法获取数据

返回一个由元组组成的可迭代对象。每个元组由DataFrame中列名和所对应的Series组成。

df4 = pd.DataFrame(np.random.randn(4,3),columns=['col1','col2','col3'])

print("df4:",df4)

i = 1

# DataFrame.iteritems()：返回一个包含列名称和内容为系列的元组

for s in df4.iteritems(): #对iteritems产生的元组进行遍历

print("第%d列数据%s"%(i,s))

i += 1

输出结果：

df: col1 col2 col3

0 -0.032271 0.420375 -0.456688

1 -0.492463 0.725930 -1.468012

2 -0.695050 0.714625 -0.331940

3 -2.942101 -0.790488 0.679380

第1列数据('col1', 0 -0.032271

1 -0.492463

2 -0.695050

3 -2.942101

Name: col1, dtype: float64)

第2列数据('col2', 0 0.420375

1 0.725930

2 0.714625

3 -0.790488

Name: col2, dtype: float64)

第3列数据('col3', 0 -0.456688

1 -1.468012

2 -0.331940

3 0.679380

Name: col3, dtype: float64)

Pandas统计函数

1） pct\_change()函数，函数将每个元素与其前一个元素进行比较，并计算变化百分比。

DataFrame.pct\_change（periods= 1，fill\_method ='pad'，limit = None，freq = None，\*\* kwargs ）：当前元素和先前元素之间的百分比变化。periods为形成百分比变化的时期，fill\_method为如何在计算百分比变化之前处理NA，limit表示停止前要填充的连续NA的数量。

s = pd.Series([1,2,3,4,5,4])

print (s.pct\_change())

df5 = pd.DataFrame(np.random.randn(5, 2))

print (df5.pct\_change())

输出结果：

0 NaN

1 1.000000

2 0.500000

3 0.333333

4 0.250000

5 -0.200000

dtype: float64

0 1

0 NaN NaN

1 -15.151902 0.174730

2 -0.746374 -1.449088

3 -3.582229 -3.165836

4 15.601150 -1.860434

2）利用Pandas计算协方差。

import pandas as pd

import numpy as np

s1 = pd.Series(np.random.randn(10))

s2 = pd.Series(np.random.randn(10))

print("s1:",s1)

print("s2:",s2)

# Series.cov（min\_periods = None）：计算列的协方差，不包括NA / null值。min\_periods表示每个列对所需的最小观测值数

print("协方差：",s1.cov(s2))

输出结果：

s1: 0 -0.928386

1 1.078359

2 -0.387015

3 -0.535971

4 -0.182057

5 -1.569763

6 -0.330704

7 -0.638071

8 -0.248482

9 1.398312

dtype: float64

s2: 0 0.385387

1 -1.136747

2 1.435437

3 -0.277588

4 -0.170578

5 -0.597523

6 0.251854

7 -0.722816

8 0.986740

9 -0.604660

dtype: float64

协方差： -0.2044109499176595

3）当应用于DataFrame时，则需要计算所有列之间的协方差(cov)值。

frame = pd.DataFrame(np.random.randn(10, 5), columns=['a', 'b', 'c', 'd', 'e'])

print(frame['a'].cov(frame['b'])) # 计算DataFrame之间的协方差

print(frame.cov())

输出结果：

-0.406796939839

a b c d e

a 0.784886 -0.406797 0.181312 0.513549 -0.597385

b -0.406797 0.987106 -0.662898 -0.492781 0.388693

c 0.181312 -0.662898 1.450012 0.484724 -0.476961

d 0.513549 -0.492781 0.484724 1.571194 -0.365274

e -0.597385 0.388693 -0.476961 -0.365274 0.785044

4）使用rank方法，返回数据的排名。

DataFrame.rank（axis = 0，method ='average'，numeric\_only = None，na\_option ='keep'，ascending = True，pct = False ）：返回数据的排名。axis选择行或列，method ='average'表示对平均值排名（可以取一下几个值'average'，'min'，'max'，'first'，'dense'），ascending = True表示降序排序，na\_option ='keep'表示将NA值保留在原来的位置。

s3 = pd.Series(np.random.randn(5), index=list('abcde')) # 生成一个Series对象

s3['d'] = s3['b'] # 另两个数据相等

print(s3.rank()) # 数据排名，根据数据大小返回名次

输出结果：

a 4.0

b 1.5 # 存在相同数据时返回名次的平均值

c 3.0

d 1.5

e 5.0

dtype: float64

Pandas中缺失数据的简单操作

1）Pandas中缺少的数据默认使用NaN填充。

df6 = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',

'h'],columns=['one', 'two', 'three'])

# 使用reindex方法设置新的索引，多出的索引对应的数据使用NaN填充

df6 = df6.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print(df6)

输出结果：

one two three

a 0.691764 -0.118095 -0.950871

b NaN NaN NaN

c -0.886898 0.053705 -1.269253

d NaN NaN NaN

e -0.344967 -0.837128 0.730831

f -1.193740 1.767796 0.888104

g NaN NaN NaN

h -0.755934 -1.331638 0.272248

2）检查数据是否存在缺失。

df7 = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',

'h'],columns=['one', 'two', 'three'])

df7 = df7.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

# reindex方法可以重置索引，多余的索引会被填充NaN

print(df7['one'].isnull()) # isnull方法可以检查数据中是否有空值

输出结果：

a False

b True

c False

d True

e False

f False

g True

h False

Name: one, dtype: bool

3）缺失数据的计算：求和数据时，NaN将被视为0，如果数据全部是NaN，那么结果将是NaN。

df8 = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',

'h'],columns=['one', 'two', 'three'])

df8 = df8.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print(df8)

print(df8['one'].sum())

输出结果：

one two three

a 0.114930 0.607963 0.273505

b NaN NaN NaN

c -0.920159 0.216739 0.843935

d NaN NaN NaN

e 1.106662 1.732285 -1.718126

f 0.490706 0.579856 0.262638

g NaN NaN NaN

h -0.533036 0.032273 0.182889

0.25910324413074204

4）用标量值替换NaN。

df9 = pd.DataFrame(np.random.randn(3, 3), index=['a', 'c', 'e'],columns=['one',

'two', 'three'])

df9 = df9.reindex(['a', 'b', 'c'])

print(df9)

print("NaN replaced with '0':")

# DataFrame.fillna（value = None，method = None，axis = None，inplace = False，limit = None，downcast = None，\*\* kwargs ）：使用指定的方法和数据填充NA / NaN值。Value表示填充数据，method表示填充方法（'backfill'，'bfill'，'pad'，'ffill'，None）。

print(df9.fillna(0)) # fillna方法可以使用指定数据来填充NaN

输出结果：

one two three

a -0.479425 -1.711840 -1.453384

b NaN NaN NaN

c -0.733606 -0.813315 0.476788

NaN replaced with '0':

one two three

a -0.479425 -1.711840 -1.453384

b 0.000000 0.000000 0.000000

c -0.733606 -0.813315 0.476788

5）删除带有NaN的数据。

DataFrame.dropna（axis = 0，how ='any'，thresh = None，subset = None，inplace = False ）：删除缺失值。How表示删除的方式（any：删除存在NA值的行或列；all：删除全部为NA的列或行）。

df10 = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f','h'],columns=['one', 'two', 'three'])

df10 = df10.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print(df10)

print("---"\*10)

print(df10.dropna())# 丢弃全部值为NaN的行，可以设置参数how=all来删除所有存在NaN值的行

输出结果：

one two three

a -2.429491 -0.597035 -1.099155

b NaN NaN NaN

c -1.241955 0.817677 -0.534722

d NaN NaN NaN

e 0.768616 -0.098987 -0.297650

f 0.600593 -1.609753 -0.733611

g NaN NaN NaN

h -0.730247 -1.315594 0.495691

------------------------------

one two three

a -2.429491 -0.597035 -1.099155

c -1.241955 0.817677 -0.534722

e 0.768616 -0.098987 -0.297650

f 0.600593 -1.609753 -0.733611

h -0.730247 -1.315594 0.495691

## 实验小结

本实验主要介绍了pandas的两种数据结构dataframe、series和它们的基本操作。

# Matplotlib

## 实验介绍

通过本实验掌握Python的2D画图工具包matplotlib的基本使用。

## 实验过程

导入绘图所需要的库

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

绘制正弦函数图

使用NumPy库生成数据，并使用matplotlib库中的plot方法对所生成的数据进行绘图。

plt.style.use("seaborn-darkgrid") # 选择图形主题（默认为全刻度）

# 共有五个主题：暗网格(darkgrid)，白网格(whitegrid)，全黑(dark)，全白(white)，全刻度

# (ticks) ,在seaborn中设置主题使用set\_style（）方法

x = np.arange(0, 3 \* np.pi, 0.1) # 生成数组x

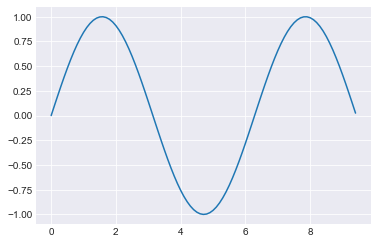
# numpy.sin(x[, out])：生成x中数据的正弦值数组

y = np.sin(x) # 对于x中的每个元素取正弦值

# matplotlib.pyplot.plot(\*args, scalex=True, scaley=True, data=None, \*\*kwargs)：绘制曲线图。\*args中包括了所需要的数据，曲线的颜色和样式等。

plt.plot(x, y) # 根据传入的数据，绘制曲线图

plt.show() #显示图形



绘制子图

matplotlib的可以把很多张图显示到一个界面：

plt.style.use("seaborn-whitegrid")

x = np.arange(0, 3 \* np.pi, 0.1)

y\_sin = np.sin(x) # 生成正弦数据

y\_cos = np.cos(x) # 生成余弦数据

# matplotlib.pyplot.subplot（\* args，\*\* kwargs ）：在当前图中添加子图。\*args是一个3位整数或三个独立的整数，用于描述子图的位置：（2,1,1）：将原图分为2行1列两个子图，后面的1表示选中第一个。

plt.subplot(2, 1, 1) # 第一个画板的第一个子图

plt.plot(x, y\_sin) # 绘制第一个子图

plt.title('Sine') # 设置图的标题

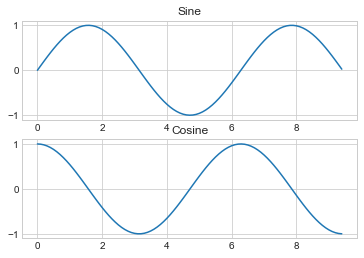
plt.subplot(2, 1, 2) # 第二个子图

plt.plot(x, y\_cos) # 绘制第二个子图

plt.title('Cosine') # 设置图的标题

plt.show()

输出结果：



图形参数

可以调整matplotlib图像中的各个参数（x，y轴、图例、颜色……）来使得图像更为符合自己的需求。

plt.style.use("seaborn-dark")

plt.figure(figsize=(8,6), dpi=80) # 创建一个8x6大小的图像, dpi=80表示分辨率每英尺80点

# 创建一个1\*1的子图

plt.subplot(111) # 等价于(1,1,1)

X = np.linspace(-np.pi, np.pi, 256,endpoint=True)

C,S = np.cos(X), np.sin(X)

# 绘制一个蓝色的，线宽为1个像素的余弦曲线, 设置图例标签Blue，linestyle表示曲线的样式

plt.plot(X, C, color="blue",linewidth=1.0,label="Blue",linestyle="--")

# 绘制一个绿色的，线宽为1个像素的正弦曲线，设置图例标签Blue，linestyle表示曲线的样式

plt.plot(X, S, color="green", linewidth=1.0, label="Green", linestyle="-.")

plt.legend() # 显示设置的图例

plt.xlim(-4.0,4.0) # 设置x轴范围

plt.xticks(np.linspace(-4,4,9,endpoint=True)) # 设置x轴刻度

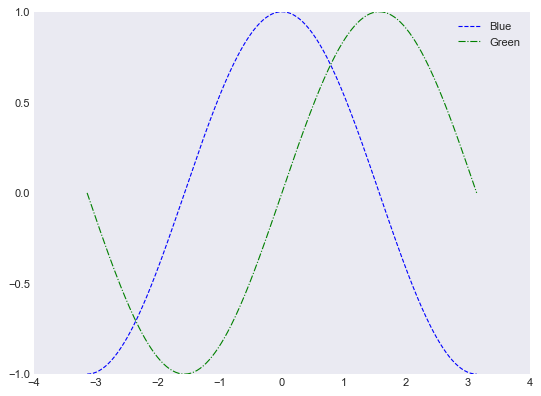
plt.ylim(-1.0,1.0) # 设置y轴范围

plt.yticks(np.linspace(-1,1,5,endpoint=True)) # 设置y轴刻度

# plt.savefig("exercice.png",dpi=72) # 保存图像，分辨率为72

plt.show()

输出结果：



调整颜色和线宽：

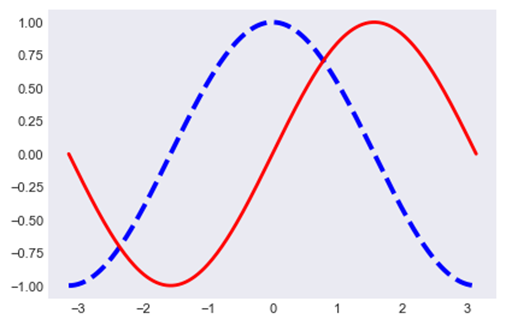
plt.figure(dpi=80)

plt.plot(X, C, color="blue", linewidth=3.5, linestyle="--")#宽度为3.5的蓝色曲线

plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")# 宽度为2.5的红色曲线

plt.show()

输出结果：



matplotlib实现散点图。

a = np.random.randint(0,20,15) # 随机生成数据

b = np.random.randint(0,20,15)

print(a)

print(b)

# matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, \*, data=None, \*\*kwargs)：绘制散点图。X、y表示xy轴的数据，s表示标量，c表示颜色，marker表示标记样式。

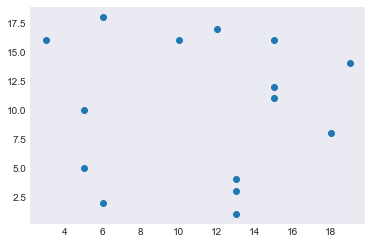
plt.scatter(a, b) # 绘制散点图

plt.show()

输出结果：

[15 18 13 15 13 5 13 3 15 10 19 6 5 6 12]

[16 8 3 11 4 5 1 16 12 16 14 18 10 2 17]



matplotlib实现柱状图

from pylab import mpl

mpl.rcParams['font.sans-serif'] = ['SimHei'] # 解决中文不显示问题

level = ['优秀', '不错', '666']

x = range(len(level)) # 横坐标

y = [1,3,2] # 纵坐标

plt.figure(dpi=100) # 创建画布

# matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, \*, align='center', data=None, \*\*kwargs)：绘制柱状图。X:横坐标；height：条形高度；width：每一个条形的宽度，color：每一个条形的颜色。

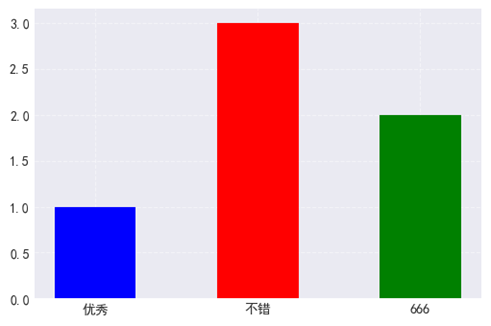
plt.bar(x, y, width=0.5, color=['b','r','g'])# 绘制柱状图

plt.xticks(x, level) # 修改x轴的刻度显示

plt.grid(linestyle="--", alpha=0.5) # 添加网格显示

plt.show()

输出结果：



matplotlib实现直方图

t = np.random.randint(0,30,90)

print(t)

plt.figure(dpi=100)

distance = 2 # 设置组距

group\_num = int((max(t) - min(t)) / distance) # 计算组数 #matplotlib.pyplot.hist(x,bins=None,range=None,density=None, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, normed=None, \*, data=None, \*\*kwargs)：绘制直方图。X：指定每个bin(箱子)分布的数据,对应x轴；bins：指定bin(箱子)的个数,也就是总共有几条条状图；normed：指定密度,也就是每个条状图的占比例比,默认为1；color ：指定条状图的颜色。

plt.hist(t,facecolor="blue", edgecolor="black", alpha=0.7) # 绘制直方图

plt.xticks(range(min(t), max(t))[::2]) # 修改x轴刻度显示

plt.grid(linestyle="--", alpha=0.5) # 添加网格显示

plt.show()

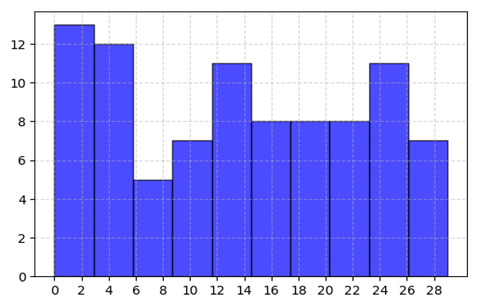
输出结果：

[28 13 7 24 4 4 20 3 0 5 4 23 26 8 26 12 25 27 17 25 3 21 13 1

25 29 11 18 27 20 1 16 12 0 23 24 19 0 12 27 9 10 10 25 14 2 14 0

3 17 17 3 27 18 27 21 2 5 17 8 13 17 1 12 5 24 15 0 7 6 14 2

22 21 10 14 3 18 24 19 0 9 16 21 22 10 25 0 19 5]



## 实验小结

本实验主要介绍了matplotlib实现简单图的形绘制，并且设置图形参数。

# Scipy

## 实验介绍

SciPy是一个开放源码的BSD许可数学，科学和工程库。SciPy库依赖于NumPy，它提供了便捷且快速的N维数组操作。

## 实验过程

### Scipy

常量

SciPy中的常量包提供了很多科学领域中的常量，如光速、π等。

#导入pi常量

from scipy.constants import \*

print("sciPy - pi = %.16f"%pi)

输出：

sciPy - pi = 3.1415927

其他常数项：

print("真空中的光速 %d"%speed\_of\_light)

print("普朗克常数 %s"%h)

print("牛顿引力系数 %s"%G)

print("电子的质量 %s"%electron\_mass)

输出：

真空中的光速 299792458

普朗克常数 6.62607015e-34

牛顿引力系数 6.6743e-11

电子的质量 9.1093837015e-31

傅里叶变换

SciPy提供了fftpack模块，包含了傅里叶变换的算法实现。

import numpy as np

#从fftpack中导入fft(快速傅里叶变化)和ifft(快速傅里叶逆变换)函数

from scipy.fftpack import fft,ifft

#创建一个随机值数组

x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])

#对数组数据进行傅里叶变换

y = fft(x)

print('fft: ')

print(y)

#快速傅里叶逆变换

yinv = ifft(y)

print('ifft: ')

print(yinv)

输出：

fft:

[ 4.5 -0.j 2.08155948-1.65109876j -1.83155948+1.60822041j

-1.83155948-1.60822041j 2.08155948+1.65109876j]

ifft:

[ 1. +0.j 2. +0.j 1. +0.j -1. +0.j 1.5+0.j]

​离散余弦变换:

#SciPy.fftpack中，提供了离散余弦变换(DCT)与离散余弦逆变换(IDCT)的实现。

import numpy as np

from scipy.fftpack import dct,idct

y = dct(np.array([4., 3., 5., 10., 5., 3.]))

print(y)

输出：

[ 60. -3.48476592 -13.85640646 11.3137085 6.

-6.31319305]

插值

数据准备：

import numpy as np

from scipy import interpolate as intp

import matplotlib.pyplot as plt

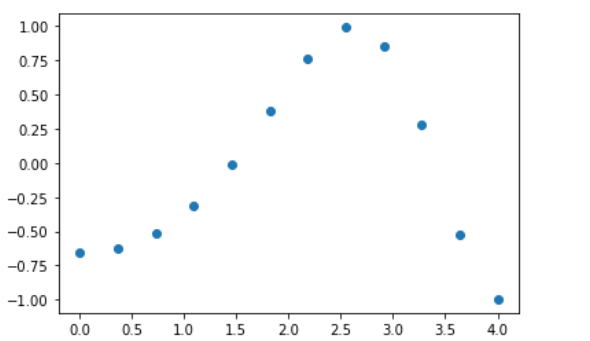
x = np.linspace(0, 4, 12)

y = np.cos(x\*\*2/3 + 4)

plt.plot(x, y,"o")

plt.show()

输出：



interp1d类可以根据输入的点，创建拟合函数。

f1 = intp.interp1d(x, y, kind = 'linear')

f2 = intp.interp1d(x, y, kind = 'cubic')

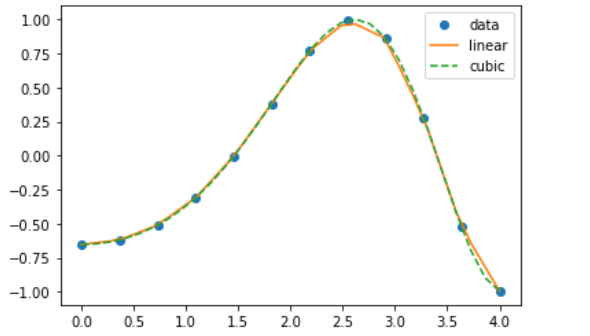
xnew = np.linspace(0, 4, 30)

plt.plot(x, y, 'o', xnew, f1(xnew), '-', xnew, f2(xnew), '--')

plt.legend(['data', 'linear', 'cubic','nearest'], loc = 'best')

plt.show()

输出：



可以通过interpolate模块中UnivariateSpline类对含有噪声的数据进行插值运算。

使用UnivariateSpline类，输入一组数据点，通过绘制一条平滑曲线来去除噪声。绘制曲线时可以设置平滑参数s，如果参数s=0，将对所有点(包括噪声)进行插值运算，也就是说s=0时不去除噪声。

import numpy as np

import matplotlib.pyplot as plt

from scipy.interpolate import UnivariateSpline

x = np.linspace(-3, 3, 50)

y = np.exp(-x\*\*2) + 0.1 \* np.random.randn(50) # 通过random方法添加噪声数据

plt.plot(x, y, 'ro', ms=5)

# 平滑参数使用默认值

spl = UnivariateSpline(x, y)

xs = np.linspace(-3, 3, 1000)

plt.plot(xs, spl(xs), 'b', lw=3) # 蓝色曲线

# 设置平滑参数

spl.set\_smoothing\_factor(0.5)

plt.plot(xs, spl(xs), 'g', lw=3) # 绿色曲线

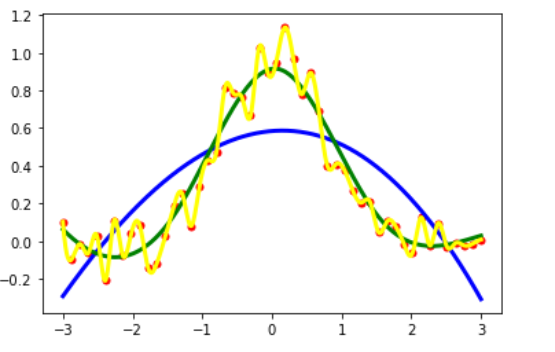
# 设置平滑参数为0

spl.set\_smoothing\_factor(0)

plt.plot(xs, spl(xs), 'yellow', lw=3) # 黄色曲线

plt.show()

输出：



SciPy.linalg

scipy.linalg.solve 函数可用于解线性方程。

# 导入scipy和numpy包

from scipy import linalg

import numpy as np

# 声明numpy数组

a = np.array([[1, 3, 5], [2, 5, 1], [2, 3, 8]])

b = np.array([10, 8, 3])

# 求解

x = linalg.solve(a, b)

# 输出解值

print (x)

输出：

[-9.28 5.16 0.76]

使用det()函数计算行列式，它接受一个矩阵作为输入，返回一个标量值，即该矩阵的行列式值。

# 行列式计算

# 声明numpy数组

A = np.array([[3,4],[7,8]])

# 计算行列式

x = linalg.det(A)

print (x)

输出：

-4.0

scipy.linalg.eig 函数可用于计算特征值与特征向量，返回特征值和特征向量。

# 声明numpy数组

A = np.array([[3,4],[7,8]])

# 求解

l, v = linalg.eig(A)

# 打印特征值

print('特征值')

print (l)

# 打印特征向量

print('特征向量')

print (v)

输出：

特征值

[-0.35234996+0.j 11.35234996+0.j]

特征向量

[[-0.76642628 -0.43192981]

[ 0.64233228 -0.90190722]]

奇异值分解SVD:

# 声明numpy数组

a = np.random.randn(3, 2) + 1.j\*np.random.randn(3, 2)

# 输出原矩阵

print('原矩阵')

print(a)

# 求解

U, s, Vh = linalg.svd(a)

# 输出结果

print('奇异值分解')

print(U, "#U")

print(Vh, "#Vh")

print(s, "#s")

输出：

原矩阵

[[ 1.51386097-0.09511335j -1.3791911 +0.7749075j ]

[-0.94634018-1.15332712j 0.20746486+0.58539069j]

[-0.42201906-1.87418033j -1.00558103-0.03405292j]]

奇异值分解

[[-0.5727813 +0.20855318j 0.11158281-0.69132048j 0.05602658+0.36730568j]

[ 0.33544796+0.39829562j 0.01989689+0.0659766j -0.67505949+0.51805516j]

[-0.01013182+0.59762845j 0.64225838+0.30394186j 0.36968399-0.03456661j]] #U

[[-0.89774342+0.j 0.4018455 +0.18049087j]

[-0.44051874+0.j -0.81893031-0.36782656j]] #Vh

[3.09614984 1.59115662] #s

## ​实验小结

本章节实验主要介绍了scipy模块的基本使用。

# Scikit-learn

## 实验介绍

机器学习是实现人工智能的重要途径，在Python中实现机器学习算法是非常容易的，我们只需要借助机器学习工具Scikit-learn，调用其中已经封装好的借口即可。本章节将通过几个小的机器学习案例来介绍Python机器学习库Scikit-learn的使用。

## 实验过程

### 特征工程

特征抽取

from sklearn.feature\_extraction import DictVectorizer

# onehot编码

data = [{'name': '张三','age':20}, {'name': '李四','age':24}, {'name': '王五','age':18}]

#实例化一个转换器类

transfer = DictVectorizer(sparse=False)

#调用fit\_transform

data = transfer.fit\_transform(data)

print("返回的结果:\n", data)

print("特征名字：\n", transfer.get\_feature\_names())

输出结果：

返回的结果:

[[20. 1. 0. 0.]

[24. 0. 1. 0.]

[18. 0. 0. 1.]]

特征名字：

['age', 'name=张三', 'name=李四', 'name=王五']

归一化

对于归一化来说：异常点会影响最后的结果。

创建数据：

import pandas as pd

l1 = [30,1,5]

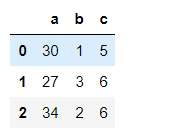
l2 = [27,3,6]

l3 = [34,2,6]

df = pd.DataFrame([l1,l2,l3], columns=["a","b","c"])

df

输出结果：



from sklearn.preprocessing import MinMaxScaler # 归一化API

transfer = MinMaxScaler(feature\_range=(0, 1))# 实例化转换器

data = transfer.fit\_transform(df[['a','b',"c"]])

data

输出结果：

array([[0.42857143, 0. , 0. ],

[0. , 1. , 1. ],

[1. , 0.5 , 1. ]])

标准化

from sklearn.preprocessing import StandardScaler

transfer = StandardScaler()

data = transfer.fit\_transform(df[['a','b',"c"]])

data

输出结果：

array([[-0.11624764, -1.22474487, -1.41421356],

[-1.16247639, 1.22474487, 0.70710678],

[ 1.27872403, 0. , 0.70710678]])

### 回归算法

线性回归

使用线性回归进行波士顿房价预测。

波士顿房价数据集是为scikit-learn中内置的数据集，共有506条数据包含了13个特征和以一个标签，每条数据包含房屋以及房屋周围的详细信息。其中包含城镇犯罪率，一氧化氮浓度，住宅平均房间数，到中心区域的加权距离以及自住房平均房价等等。

导入所需工具：

from sklearn.datasets import load\_boston # 导入数据波士顿房价

from sklearn.linear\_model import SGDRegressor# 线性回归

from sklearn.model\_selection import train\_test\_split# 划分数据集

from sklearn.preprocessing import StandardScaler# 数据标准化

from sklearn.metrics import mean\_squared\_error # 均方误差

查看数据的一些属性：

boston = load\_boston() # 导入加载的数据集

boston = load\_boston()

print("数据维度：",boston.data.shape)# 查看数据维度

print("房价数据：",boston.data)# 查看数据

print("特征：",boston.feature\_names)# 查看数据的特征名称

print("标签：",boston.target)# 查看标签数据

输出结果：

数据维度： (506, 13)

房价数据： [[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00]

[2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00]

[2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00]

...

[6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00]

[1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00]

[4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]

特征： ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'

'B' 'LSTAT']

标签： [24. 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15. 18.9 21.7 20.4

18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8

18.4 21. 12.7 14.5 13.2 13.1 13.5 18.9 20. 21. 24.7 30.8 34.9 26.6

25.3 24.7 21.2 19.3 20. 16.6 14.4 19.4 19.7 20.5 25. 23.4 18.9 35.4

24.7 31.6 23.3 19.6 18.7 16. 22.2 25. 33. 23.5 19.4 22. 17.4 20.9

24.2 21.7 22.8 23.4 24.1 21.4 20. 20.8 21.2 20.3 28. 23.9 24.8 22.9

23.9 26.6 22.5 22.2 23.6 28.7 22.6 22. 22.9 25. 20.6 28.4 21.4 38.7

43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8

18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22. 20.3 20.5 17.3 18.8 21.4

15.7 16.2 18. 14.3 19.2 19.6 23. 18.4 15.6 18.1 17.4 17.1 13.3 17.8

14. 14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4

17. 15.6 13.1 41.3 24.3 23.3 27. 50. 50. 50. 22.7 25. 50. 23.8

23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2

37.9 32.5 26.4 29.6 50. 32. 29.8 34.9 37. 30.5 36.4 31.1 29.1 50.

33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50. 22.6 24.4 22.5 24.4 20.

21.7 19.3 22.4 28.1 23.7 25. 23.3 28.7 21.5 23. 26.7 21.7 27.5 30.1

44.8 50. 37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29. 24. 25.1 31.5

23.7 23.3 22. 20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8

29.6 42.8 21.9 20.9 44. 50. 36. 30.1 33.8 43.1 48.8 31. 36.5 22.8

30.7 50. 43.5 20.7 21.1 25.2 24.4 35.2 32.4 32. 33.2 33.1 29.1 35.1

45.4 35.4 46. 50. 32.2 22. 20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9

21.7 28.6 27.1 20.3 22.5 29. 24.8 22. 26.4 33.1 36.1 28.4 33.4 28.2

22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21. 23.8 23.1

20.4 18.5 25. 24.6 23. 22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1

19.5 18.5 20.6 19. 18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6

22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25. 19.9 20.8 16.8

21.9 27.5 21.9 23.1 50. 50. 50. 50. 50. 13.8 13.8 15. 13.9 13.3

13.1 10.2 10.4 10.9 11.3 12.3 8.8 7.2 10.5 7.4 10.2 11.5 15.1 23.2

9.7 13.8 12.7 13.1 12.5 8.5 5. 6.3 5.6 7.2 12.1 8.3 8.5 5.

11.9 27.9 17.2 27.5 15. 17.2 17.9 16.3 7. 7.2 7.5 10.4 8.8 8.4

16.7 14.2 20.8 13.4 11.7 8.3 10.2 10.9 11. 9.5 14.5 14.1 16.1 14.3

11.7 13.4 9.6 8.7 8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6

14.1 13. 13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20. 16.4 17.7

19.5 20.2 21.4 19.9 19. 19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3

16.7 12. 14.6 21.4 23. 23.7 25. 21.8 20.6 21.2 19.1 20.6 15.2 7.

8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9

22. 11.9]

划分数据集：

x\_train, x\_test, y\_train, y\_test = train\_test\_split(boston.data, boston.target,test\_size=0.2, random\_state=6)

print("训练集：",x\_train)

print("训练集维度：",x\_train.shape)

print("测试集：",x\_test)

print("测试集维度：",x\_test.shape)

输出结果：

训练集： [[2.53870e-01 0.00000e+00 6.91000e+00 ... 1.79000e+01 3.96900e+02

3.08100e+01]

[7.52601e+00 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.04210e+02

1.93100e+01]

[3.30600e-02 0.00000e+00 5.19000e+00 ... 2.02000e+01 3.96140e+02

8.51000e+00]

...

[4.12380e-01 0.00000e+00 6.20000e+00 ... 1.74000e+01 3.72080e+02

6.36000e+00]

[3.44500e-02 8.25000e+01 2.03000e+00 ... 1.47000e+01 3.93770e+02

7.43000e+00]

[1.33598e+01 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.96900e+02

1.63500e+01]]

训练集维度： (404, 13)

测试集： [[1.96091e+01 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.96900e+02

1.34400e+01]

[1.87000e-02 8.50000e+01 4.15000e+00 ... 1.79000e+01 3.92430e+02

6.36000e+00]

[6.14700e-01 0.00000e+00 6.20000e+00 ... 1.74000e+01 3.96900e+02

7.60000e+00]

...

[1.78667e+01 0.00000e+00 1.81000e+01 ... 2.02000e+01 3.93740e+02

2.17800e+01]

[1.05393e+00 0.00000e+00 8.14000e+00 ... 2.10000e+01 3.86850e+02

6.58000e+00]

[1.98020e-01 0.00000e+00 1.05900e+01 ... 1.86000e+01 3.93630e+02

9.47000e+00]]

测试集维度： (102, 13)

数据标准化。

transfer = StandardScaler()# 实例化标准化对象

x\_train = transfer.fit\_transform(x\_train)# 将数据进行标准化

x\_test = transfer.transform(x\_test)

print("标准化：",x\_train)# 数据在标准化后数值发生了改变，但是数据维度没有变化。

print('标准化后的维度：',x\_train.shape)

输出结果：

标准化： [[-0.39262879 -0.48699244 -0.58398492 ... -0.23241985 0.43823462

2.60935849]

[ 0.49588716 -0.48699244 1.0378004 ... 0.82415806 -0.56005456

0.97851821]

[-0.41960753 -0.48699244 -0.83326738 ... 0.82415806 0.43004927

-0.55305353]

...

[-0.37326191 -0.48699244 -0.6868864 ... -0.4621107 0.17091843

-0.85794976]

[-0.4194377 2.9643063 -1.29125144 ... -1.70244128 0.40452391

-0.70621071]

[ 1.20866434 -0.48699244 1.0378004 ... 0.82415806 0.43823462

0.5587541 ]]

标准化后的维度： (404, 13)

实现线性回归算法。

estimator = SGDRegressor()# 线性回归

estimator.fit(x\_train, y\_train)# 使用fit方法填充数据进行训练

预测：

y\_predict = estimator.predict(x\_test)# predict方法进行测试

print(y\_predict)

输出结果：

[25.607803 25.10588403 28.45554286 23.49231839 19.85942653 19.80180393

15.99055852 42.62340201 21.48208447 20.69415769 24.57293165 12.33931696

31.98415943 31.89546681 33.21128973 19.65171805 13.58355015 28.09264257

17.04285943 16.29096574 16.40691677 28.09228097 23.08342694 19.14053746

19.58859606 19.89834326 20.95535376 28.48600746 19.90188641 24.57526155

24.04757837 22.32724439 31.76457359 37.03606573 19.24150227 23.21074541

15.09067588 20.21735263 17.97933219 3.73937273 11.79576307 11.76535647

13.94773717 12.94361242 20.79451245 24.53029657 31.90397554 22.64792077

19.45973628 18.10537068 20.92238088 14.79247057 23.22643436 19.96984143

35.56419239 26.02450385 13.10329184 29.19950228 17.71056396 20.04543381

23.4666464 22.75070692 12.56614955 28.26103663 17.0949352 15.34893491

32.05586083 14.81292302 25.88568088 33.92786038 9.48039192 21.57257841

13.89542731 24.70383567 30.59989092 22.49239627 16.49428623 39.57352352

13.70417373 20.85291033 20.49957225 13.33464936 -0.24163188 34.2294614

17.52567634 34.31478255 11.8586464 18.20176197 14.14902915 37.87194414

12.67641575 2.70709017 15.4221406 27.00426598 20.69144517 17.54440457

26.55480213 18.37210151 25.77213609 16.65240775 20.84829901 23.5904609 ]

使sklearn中的均方误差函数计算误差，在回归实验中MSE较为常用。

error = mean\_squared\_error(y\_test, y\_predict)# 均方误差

print(error)

输出结果：

26.496967227397487

可视化结果进行查看。

# 使用matplotlib进行可视化来查看预测结和只是结果的差异性

import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))

plt.xlabel("x",fontsize=14)

plt.ylabel("y", fontsize=14)

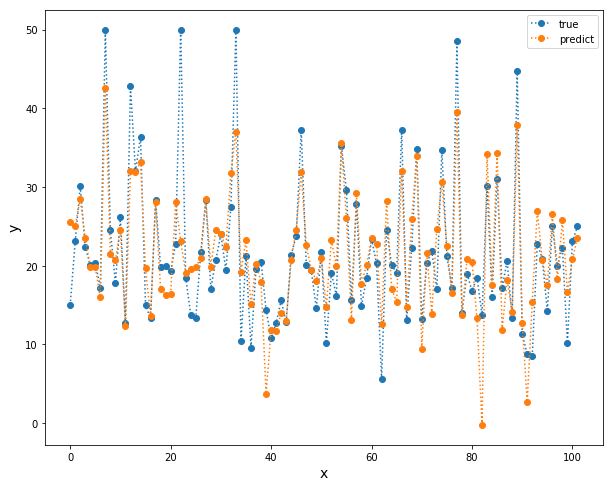
plt.plot([i for i in range(len(y\_test))], y\_test,linestyle=':', marker='o' ,label="true")

plt.plot([i for i in range(len(y\_test))], y\_predict, linestyle=':',marker='o' ,label="predict")

plt.legend()

plt.show()

输出结果：



### 分类算法

逻辑回归

使用逻辑回归来进行癌症分类预测。

Breast-Cancer数据集（数据地址：https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/）：有11个列，第1个列为id号，第2-10列为特征，11列为标签（2为良性、4为恶性）。

* 数据集中共有699条信息；
* 16处缺失值，缺失值使用"?"表示；
* 良性数据有458条，恶性数据有241条

由于实验中所用到的数据是开源数据，这些数据已经被专业人士处理过了，所以我们不需要做过的处理（下同）。

# train\_test\_split用来划分数据集

from sklearn.model\_selection import train\_test\_split

# 逻辑回归

from sklearn.linear\_model import LogisticRegression

column\_name = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',

'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',

'Normal Nucleoli', 'Mitoses', 'Class']

#若在华为云上实验，请手动下载数据集后上传至Modelarts。

data = pd.read\_csv(r"https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data",

names=column\_name)

# 删除缺失值

data = data.replace(to\_replace='?', value=np.nan)

data = data.dropna()

# 取出特征值

x = data[column\_name[1:10]]

y = data[column\_name[10]]

# 分割数据集 测试集占比30%

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3)

# 进行标准化

std = StandardScaler()

x\_train = std.fit\_transform(x\_train)

x\_test = std.transform(x\_test)

# 使用逻辑回归

lr = LogisticRegression()

lr.fit(x\_train, y\_train)

print("得出来的权重：", lr.coef\_)

# 预测类别

print("预测的类别：", lr.predict(x\_test))

# 得出准确率

print("预测的准确率:", lr.score(x\_test, y\_test))

输出结果：

得出来的权重： [[1.2399326 0.32716634 0.87241194 0.63742043 0.25401877 1.03994695

0.77241605 0.63150512 0.65476603]]

预测的类别： [4 2 2 2 2 4 2 2 4 2 2 2 2 2 2 4 2 4 2 2 2 2 2 4 2 4 2 2 2 4 2 4 2 4 4 4 4

2 4 2 2 2 2 4 2 4 2 4 2 2 2 4 4 2 4 4 2 4 2 2 4 2 4 2 2 4 4 4 2 2 2 4 2 2

2 2 2 4 2 2 4 2 4 4 2 4 2 2 2 4 4 2 4 4 4 2 2 4 2 2 2 4 4 4 2 2 2 2 4 2 2

4 2 2 4 2 2 2 2 2 2 2 2 4 2 4 2 4 4 4 2 4 2 2 2 4 4 4 2 2 2 2 2 2 2 4 2 2

2 2 4 2 4 2 4 4 2 4 2 2 2 2 4 2 4 2 2 2 2 2 4 2 2 2 4 4 2 2 4 2 2 2 4 4 2

4 2 2 2 4 4 2 2 2 2 2 4 2 4 4 2 2 4 2 2]

预测的准确率: 0.9804878048780488

KNN

使用KNN算法实现鸢尾花种类预测。

鸢尾花数据集：数据集内包含 3 类共 150 条记录，每类各 50 个数据，每条记录都有 4 项特征：花萼长度、花萼宽度、花瓣长度、花瓣宽度，可以通过这4个特征预测鸢尾花卉属于（iris-setosa, iris-versicolour, iris-virginica）中的哪一品种。

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

iris = load\_iris()

# x\_train,x\_test,y\_train,y\_test为训练集特征值、测试集特征值、训练集目标值、测试集目标值

x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2, random\_state=22)

transfer = StandardScaler()

x\_train = transfer.fit\_transform(x\_train)

x\_test = transfer.transform(x\_test)

# 实例化KNN分类器

estimator = KNeighborsClassifier(n\_neighbors=9)

estimator.fit(x\_train, y\_train)

# 模型评估

y\_predict = estimator.predict(x\_test)

print("预测结果为:\n", y\_predict)

print("比对真实值和预测值：\n", y\_predict == y\_test)

score = estimator.score(x\_test, y\_test)

print("准确率为：\n", score)

输出结果：

预测结果为:

[0 2 1 2 1 1 1 1 1 0 2 1 2 2 0 2 1 1 1 1 0 2 0 1 2 0 2 2 2 2]

比对真实值和预测值：

[ True True True True True True True False True True True True

True True True True True True False True True True True True

True True True True True True]

准确率为：

0.9333333333333333

决策树

使用决策树算法实现泰坦尼克号乘客生存预测。

泰塔尼克号数据地址：<http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt>。

数据中共有12个特诊，此次试验我们选取其中的‘pclass', 'age', 'sex'三列数据作为实验中的特征。

from sklearn.tree import DecisionTreeClassifier

titan = pd.read\_csv("http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt")

x = titan[['pclass', 'age', 'sex']]

y = titan['survived']

# 缺失值需要处理，将特征当中有类别的这些特征进行字典特征抽取

x['age'].fillna(x['age'].mean(), inplace=True)

# 对于x转换成字典数据x.to\_dict(orient="records")

# [{"pclass": "1st", "age": 29.00, "sex": "female"}, {}]

dict = DictVectorizer(sparse=False)

x = dict.fit\_transform(x.to\_dict(orient="records"))

print(dict.get\_feature\_names())

print(x)

# 分割训练集合测试集

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3)

# 进行决策树的建立和预测

# 指定树的深度大小为5

dc = DecisionTreeClassifier(criterion='entropy', max\_depth=5)

dc.fit(x\_train, y\_train)

print("预测的准确率为：", dc.score(x\_test, y\_test))

输出结果：

['age', 'pclass=1st', 'pclass=2nd', 'pclass=3rd', 'sex=female', 'sex=male']

[[29. 1. 0. 0. 1. 0. ]

[ 2. 1. 0. 0. 1. 0. ]

[30. 1. 0. 0. 0. 1. ]

...

[31.19418104 0. 0. 1. 0. 1. ]

[31.19418104 0. 0. 1. 1. 0. ]

[31.19418104 0. 0. 1. 0. 1. ]]

预测的准确率为： 0.8401015228426396

### 聚类算法

K-means

使用k-means聚类算法实现鸢尾花的聚类操作。

k-means对于大型数据集也是简单高效、时间复杂度、空间复杂度低。 最重要是数据集大时结果容易局部最优；需要预先设定K值，对最先的K个点选取很敏感；对噪声和离群值非常敏感。

import matplotlib.pyplot as plt

import numpy as np

from sklearn.cluster import KMeans

from sklearn import datasets

iris = datasets.load\_iris()

X = iris.data[:, :4] # #表示我们取特征空间中的4个维度

print(X.shape)

# 绘制数据分布图

plt.scatter(X[:, 0], X[:, 1], c="red", marker='o', label='data')

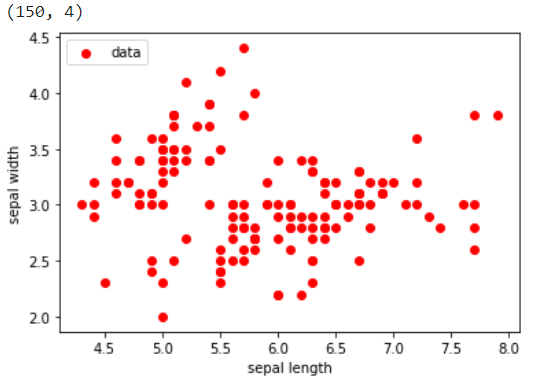
plt.xlabel('sepal length')

plt.ylabel('sepal width')

plt.legend(loc=2)

plt.show()

输出结果：



estimator = KMeans(n\_clusters=3) # 构造聚类器

estimator.fit(X) # 聚类

label\_pred = estimator.labels\_ # 获取聚类标签

# 绘制k-means结果

x0 = X[label\_pred == 0]

x1 = X[label\_pred == 1]

x2 = X[label\_pred == 2]

plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o', label='label0')

plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='\*', label='label1')

plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker='+', label='label2')

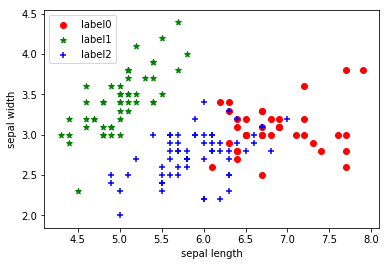
plt.xlabel('sepal length')

plt.ylabel('sepal width')

plt.legend(loc=2)

plt.show()

输出结果：



DBSCAN

对于同样的鸢尾花数据使用DBSCAN进行聚类操作。DBSCAN对噪声不敏感；能发现任意形状的聚类。 但是聚类的结果与参数有很大的关系；DBSCAN用固定参数识别聚类，但当聚类的稀疏程度不同时，相同的判定标准可能会破坏聚类的自然结构，即较稀的聚类会被划分为多个类或密度较大且离得较近的类会被合并成一个聚类。

DBSCAN中有两个非常中要的参数eps和min\_samples这两个参数表示数据的稠密性。当min\_samples增加 或者 eps 减小的时候，意味着一个簇分类有更大的密度要求。

estimator = DBSCAN(eps = 0.4,min\_samples = 4) # 构造聚类器

estimator.fit(X) # 聚类

label\_pred = estimator.labels\_ # 获取聚类标签

# 绘制k-means结果

x0 = X[label\_pred == 0]

x1 = X[label\_pred == 1]

x2 = X[label\_pred == 2]

plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o', label='label0')

plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='\*', label='label1')

plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker='+', label='label2')

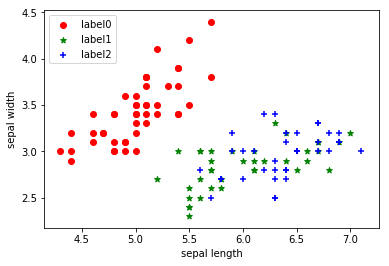
plt.xlabel('sepal length')

plt.ylabel('sepal width')

plt.legend(loc=2)

plt.show()

输出结果：



## 实验小结

本实验主要介绍了Python机器学习工具sklearn的使用，通过sklearn来实现常见的特征工程操作和集中简单的机器学习算法，通过回归、分类和聚类算法的实现来进一步了解sklearn的使用。

# MindSpore

## 实验介绍

Mindspore是最佳匹配昇腾AI处理器算力的全场景深度学习框架，为数据科学家和算法工程师提供设计友好、运行高效的开发体验，推动人工智能软硬件应用生态繁荣发展。本章将通过手写数字识别的案例讲述华为AI开发框架Mindspore的基本使用。

实验

## 资源准备

下载并安装MindSpore：



或者直接使用ModelArts中MndSpore内核。

## 实验过程

实验流程：

1、处理需要的数据集，这里使用了MNIST数据集。

2、定义一个网络，这里我们使用LeNet网络。

3、定义损失函数和优化器。

4、加载数据集并进行训练，训练完成后，查看结果及保存模型文件。

5、加载保存的模型，进行推理。

6、验证模型，加载测试数据集和训练后的模型，验证结果精度。

### 手写数字识别

获取数据集

直接执行下面代码，会自动进行训练集的下载与解压，但是整个过程根据网络好坏情况会需要花费几分钟时间。

# Network request module, data download module, decompression module

import urllib.request

from urllib.parse import urlparse

import gzip

import os

def unzipfile(gzip\_path):

"""unzip dataset file

Args:

gzip\_path: dataset file path

"""

open\_file = open(gzip\_path.replace('.gz',''), 'wb')

gz\_file = gzip.GzipFile(gzip\_path)

open\_file.write(gz\_file.read())

gz\_file.close()

def download\_dataset():

"""Download the dataset from http://yann.lecun.com/exdb/mnist/."""

print("\*\*\*\*\*\*Downloading the MNIST dataset\*\*\*\*\*\*")

train\_path = "./MNIST\_Data/train/"

test\_path = "./MNIST\_Data/test/"

train\_path\_check = os.path.exists(train\_path)

test\_path\_check = os.path.exists(test\_path)

if train\_path\_check == False and test\_path\_check == False:

os.makedirs(train\_path)

os.makedirs(test\_path)

train\_url = {"https://share-whc.obs.cn-north-4.myhuaweicloud.com:443/mnist-data/train-images-idx3-ubyte.gz", "https://share-whc.obs.cn-north-4.myhuaweicloud.com:443/mnist-data/train-labels-idx1-ubyte.gz"}

test\_url = {"https://share-whc.obs.cn-north-4.myhuaweicloud.com:443/mnist-data/t10k-images-idx3-ubyte.gz", "https://share-whc.obs.cn-north-4.myhuaweicloud.com:443/mnist-data/t10k-labels-idx1-ubyte.gz"}

for url in train\_url:

url\_parse = urlparse(url)

# split the file name from url

file\_name = os.path.join(train\_path,url\_parse.path.split('/')[-1])

if not os.path.exists(file\_name.replace('.gz', '')):

file = urllib.request.urlretrieve(url, file\_name)

unzipfile(file\_name)

os.remove(file\_name)

for url in test\_url:

url\_parse = urlparse(url)

# split the file name from url

file\_name = os.path.join(test\_path,url\_parse.path.split('/')[-1])

if not os.path.exists(file\_name.replace('.gz', '')):

file = urllib.request.urlretrieve(url, file\_name)

unzipfile(file\_name)

os.remove(file\_name)

download\_dataset()

处理MNIST数据集

由于我们后面会采用LeNet这样的卷积神经网络对数据集进行训练，而采用LeNet在训练数据时，对数据格式是有所要求的，所以接下来的工作需要我们先查看数据集内的数据是什么样的，这样才能构造一个针对性的数据转换函数，将数据集数据转换成符合训练要求的数据形式。

查看原始数据集数据：

from mindspore import context

import matplotlib.pyplot as plt

import matplotlib

import numpy as np

import mindspore.dataset as ds

context.set\_context(mode=context.GRAPH\_MODE, device\_target="CPU") # Windows version, set to use CPU for graph calculation

train\_data\_path = "./MNIST\_Data/train"

test\_data\_path = "./MNIST\_Data/test"

mnist\_ds = ds.MnistDataset(train\_data\_path) # Load training dataset

print('The type of mnist\_ds:', type(mnist\_ds))

print("Number of pictures contained in the mnist\_ds：",mnist\_ds.get\_dataset\_size()) # 60000 pictures in total

dic\_ds = mnist\_ds.create\_dict\_iterator() # Convert dataset to dictionary type

item = dic\_ds.\_\_next\_\_()

img = item["image"].asnumpy()

label = item["label"].asnumpy()

print("The item of mnist\_ds:", item.keys()) # Take a single data to view the data structure, including two keys, image and label

print("Tensor of image in item:", img.shape) # View the tensor of image (28,28,1)

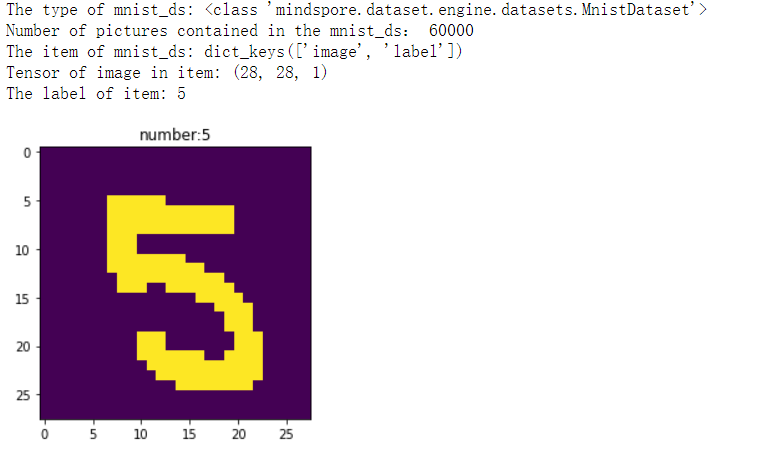
print("The label of item:", label)

plt.imshow(np.squeeze(img))

plt.title("number:%s"% item["label"])

plt.show()

输出：



从上面的运行情况我们可以看到,训练数据集train-images-idx3-ubyte和train-labels-idx1-ubyte对应的是6万张图片和6万个数字下标，载入数据后经过create\_dict\_iterator()转换字典型的数据集，取其中的一个数据查看，这是一个key为image和label的字典，其中的image的张量(高度28，宽度28，通道1)和label为对应图片的数字。

数据处理

定义数据集及数据操作

我们定义一个函数create\_dataset()来创建数据集。在这个函数中，我们定义好需要进行的数据增强和处理操作：

1、定义数据集。

2、定义进行数据增强和处理所需要的一些参数。

3、根据参数，生成对应的数据增强操作。

4、使用map()映射函数，将数据操作应用到数据集。

5、对生成的数据集进行处理。

# Data processing module

import mindspore.dataset.vision.c\_transforms as CV

import mindspore.dataset.transforms.c\_transforms as C

from mindspore.dataset.vision import Inter

from mindspore.common import dtype as mstype

def create\_dataset(data\_path, batch\_size=32, repeat\_size=1,

num\_parallel\_workers=1):

""" create dataset for train or test

Args:

data\_path: Data path

batch\_size: The number of data records in each group

repeat\_size: The number of replicated data records

num\_parallel\_workers: The number of parallel workers

"""

# define dataset

mnist\_ds = ds.MnistDataset(data\_path)

# Define some parameters needed for data enhancement and rough justification

resize\_height, resize\_width = 32, 32

rescale = 1.0 / 255.0

shift = 0.0

rescale\_nml = 1 / 0.3081

shift\_nml = -1 \* 0.1307 / 0.3081

# According to the parameters, generate the corresponding data enhancement method

resize\_op = CV.Resize((resize\_height, resize\_width), interpolation=Inter.LINEAR) # Resize images to (32, 32) by bilinear interpolation

rescale\_nml\_op = CV.Rescale(rescale\_nml, shift\_nml) # normalize images

rescale\_op = CV.Rescale(rescale, shift) # rescale images

hwc2chw\_op = CV.HWC2CHW() # change shape from (height, width, channel) to (channel, height, width) to fit network.

type\_cast\_op = C.TypeCast(mstype.int32) # change data type of label to int32 to fit network

# Using map () to apply operations to a dataset

mnist\_ds = mnist\_ds.map(input\_columns="label", operations=type\_cast\_op, num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(input\_columns="image", operations=resize\_op, num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(input\_columns="image", operations=rescale\_op, num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(input\_columns="image", operations=rescale\_nml\_op, num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(input\_columns="image", operations=hwc2chw\_op, num\_parallel\_workers=num\_parallel\_workers)

# Process the generated dataset

buffer\_size = 10000

mnist\_ds = mnist\_ds.shuffle(buffer\_size=buffer\_size) # 10000 as in LeNet train script

mnist\_ds = mnist\_ds.batch(batch\_size, drop\_remainder=True)

mnist\_ds = mnist\_ds.repeat(repeat\_size)

return mnist\_ds

batch\_size：每组包含的数据个数，现设置每组包含32个数据。

repeat\_size：数据集复制的数量。

先进行shuffle、batch操作，再进行repeat操作，这样能保证1个epoch内数据不重复。

查看数据集内包含多少组数据。

datas = create\_dataset(train\_data\_path) # Process the train dataset

print('Number of groups in the dataset:', datas.get\_dataset\_size()) # Number of query dataset groups

输出：

Number of groups in the dataset: 1875

取出其中一组数据，查看包含的key，图片数据的张量，以及下标labels的值。

data = datas.create\_dict\_iterator().\_\_next\_\_() # Take a set of datasets

print(data.keys())

images = data["image"].asnumpy() # Take out the image data in this dataset

labels = data["label"].asnumpy() # Take out the label (subscript) of this data set

print('Tensor of image:', images.shape) # Query the tensor of images in each dataset (32,1,32,32)

print('labels:', labels)

输出：

dict\_keys(['label', 'image'])

Tensor of image: (32, 1, 32, 32)

labels: [8 2 8 5 1 0 0 2 6 0 7 6 7 9 3 3 8 9 5 6 6 5 5 5 9 5 1 8 9 7 2 7]

查看image的图像和下标对应的值。

count = 1

for i in images:

plt.subplot(4, 8, count)

plt.imshow(np.squeeze(i))

plt.title('num:%s'%labels[count-1])

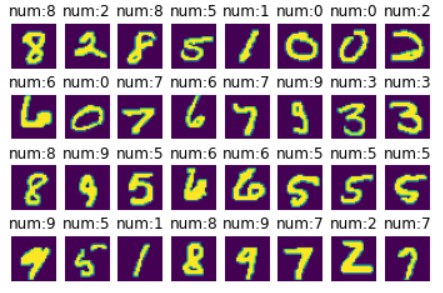
plt.xticks([])

count += 1

plt.axis("off")

plt.show() # Print a total of 32 pictures in the group

输出：



构造神经网络

在对手写字体识别上，通常采用卷积神经网络架构（CNN）进行学习预测，最经典的属1998年由Yann LeCun创建的LeNet5架构，  
其中分为：  
1、输入层；  
2、卷积层C1；  
3、池化层S2；  
4、卷积层C3；  
5、池化层S4；  
6、全连接F6；  
7、全连接；  
8、全连接OUTPUT。

在构建LeNet5前，我们需要对全连接层以及卷积层进行初始化。

TruncatedNormal：参数初始化方法，MindSpore支持TruncatedNormal、Normal、Uniform等多种参数初始化方法，具体可以参考MindSpore API的mindspore.common.initializer模块说明。

import mindspore.nn as nn

from mindspore.common.initializer import TruncatedNormal

# Initialize 2D convolution function

def conv(in\_channels, out\_channels, kernel\_size, stride=1, padding=0):

"""Conv layer weight initial."""

weight = weight\_variable()

return nn.Conv2d(in\_channels, out\_channels,

kernel\_size=kernel\_size, stride=stride, padding=padding,

weight\_init=weight, has\_bias=False, pad\_mode="valid")

# Initialize full connection layer

def fc\_with\_initialize(input\_channels, out\_channels):

"""Fc layer weight initial."""

weight = weight\_variable()

bias = weight\_variable()

return nn.Dense(input\_channels, out\_channels, weight, bias)

# Set truncated normal distribution

def weight\_variable():

"""Weight initial."""

return TruncatedNormal(0.02)

使用MindSpore定义神经网络需要继承mindspore.nn.cell.Cell，Cell是所有神经网络（Conv2d等）的基类。

神经网络的各层需要预先在\_\_init\_\_()方法中定义，然后通过定义construct()方法来完成神经网络的前向构造，按照LeNet5的网络结构，定义网络各层如下：

class LeNet5(nn.Cell):

"""Lenet network structure."""

# define the operator required

def \_\_init\_\_(self):

super(LeNet5, self).\_\_init\_\_()

self.batch\_size = 32 # 32 pictures in each group

self.conv1 = conv(1, 6, 5) # Convolution layer 1, 1 channel input (1 Figure), 6 channel output (6 figures), convolution core 5 \* 5

self.conv2 = conv(6, 16, 5) # Convolution layer 2,6-channel input, 16 channel output, convolution kernel 5 \* 5

self.fc1 = fc\_with\_initialize(16 \* 5 \* 5, 120)

self.fc2 = fc\_with\_initialize(120, 84)

self.fc3 = fc\_with\_initialize(84, 10)

self.relu = nn.ReLU()

self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

# use the preceding operators to construct networks

def construct(self, x):

x = self.conv1(x) # 1\*32\*32-->6\*28\*28

x = self.relu(x) # 6\*28\*28-->6\*14\*14

x = self.max\_pool2d(x) # Pool layer

x = self.conv2(x) # Convolution layer

x = self.relu(x) # Function excitation layer

x = self.max\_pool2d(x) # Pool layer

x = self.flatten(x) # Dimensionality reduction

x = self.fc1(x) # Full connection

x = self.relu(x) # Function excitation layer

x = self.fc2(x) # Full connection

x = self.relu(x) # Function excitation layer

x = self.fc3(x) # Full connection

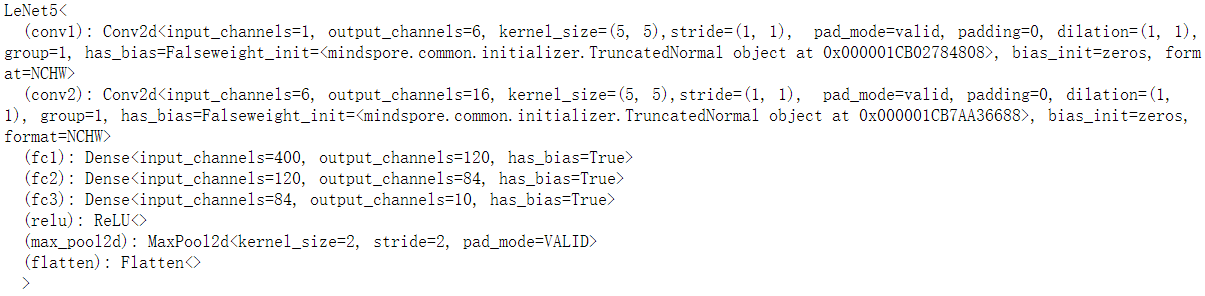
return x

构建完成后，我们将LeNet5的整体参数打印出来查看一下。

network = LeNet5()

print(network)

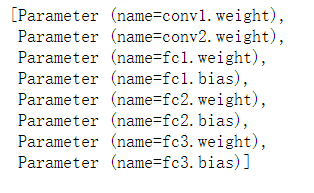
输出：



param = network.trainable\_params()

param

输出：



搭建训练网络并进行训练

构建完成神经网络后，就可以着手进行训练网络的构建，模型训练函数为Model.train(),参数主要包含:

1、圈数epoch size（每圈需要遍历完成1875组图片）;

2、数据集ds\_train;

3、回调函数callbacks包含ModelCheckpoint、LossMonitor、SummaryStepckpoint\_cb,Callback模型检测参数;

4、底层数据通道dataset\_sink\_mode，此参数默认True需设置成False，因为此功能不支持CPU模式（当前环境为CPU）。

# Training and testing related modules

import argparse

from mindspore import Tensor

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor,Callback

from mindspore.train import Model

from mindspore.nn.metrics import Accuracy

from mindspore.nn.loss import SoftmaxCrossEntropyWithLogits

def train\_net(model, epoch\_size, mnist\_path, repeat\_size, ckpoint\_cb, step\_loss\_info):

"""Define the training method."""

print("============== Starting Training ==============")

# load training dataset

ds\_train = create\_dataset(os.path.join(mnist\_path, "train"), 32, repeat\_size)

model.train(epoch\_size, ds\_train, callbacks=[ckpoint\_cb, LossMonitor(), step\_loss\_info], dataset\_sink\_mode=False)

自定义一个存储每一步训练的step和对应loss值的类Step\_loss\_info()，并继承了Callback类，可以自定义训练过程中的处理措施，非常方便，等训练完成后，可将数据绘图查看loss的变化情况。

# Custom callback function

class Step\_loss\_info(Callback):

def step\_end(self, run\_context):

cb\_params = run\_context.original\_args()

# step\_ Loss dictionary for saving loss value and step number information

step\_loss["loss\_value"].append(str(cb\_params.net\_outputs))

step\_loss["step"].append(str(cb\_params.cur\_step\_num))

定义损失函数及优化器

MindSpore支持的损失函数有SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss等。这里使用SoftmaxCrossEntropyWithLogits损失函数。

import os

if os.name == "nt":

os.system('del/f/s/q \*.ckpt \*.meta')# Clean up old run files before in Windows

else:

os.system('rm -f \*.ckpt \*.meta \*.pb')# Clean up old run files before in Linux

lr = 0.01 # learning rate

momentum = 0.9 #

# create the network

network = LeNet5()

# define the optimizer

net\_opt = nn.Momentum(network.trainable\_params(), lr, momentum)

# define the loss function

net\_loss = SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# define the model

model = Model(network, net\_loss, net\_opt, metrics={"Accuracy": Accuracy()} )

epoch\_size = 1

mnist\_path = "./MNIST\_Data"

config\_ck = CheckpointConfig(save\_checkpoint\_steps=125, keep\_checkpoint\_max=16)

# save the network model and parameters for subsequence fine-tuning

ckpoint\_cb = ModelCheckpoint(prefix="checkpoint\_lenet", config=config\_ck)

# group layers into an object with training and evaluation features

step\_loss = {"step": [], "loss\_value": []}

# step\_ Loss dictionary for saving loss value and step number information

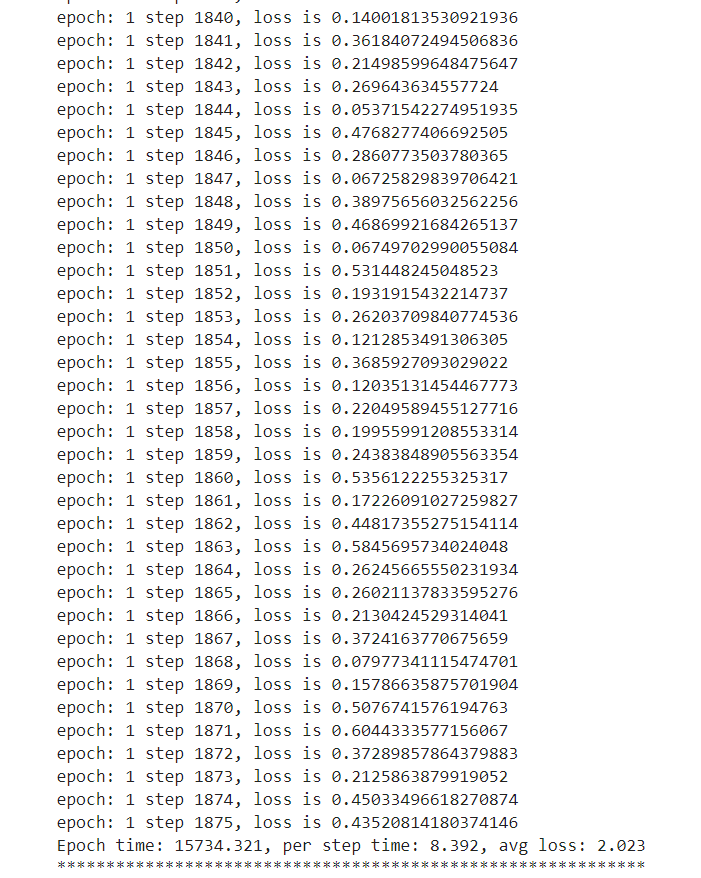
step\_loss\_info = Step\_loss\_info()

# save the steps and loss value

repeat\_size = 1

train\_net(model, epoch\_size, mnist\_path, repeat\_size, ckpoint\_cb, step\_loss\_info)

输出：



训练完成后，能在Jupyter的工作路径上生成多个模型文件，名称具体含义checkpoint\_{网络名称}-{第几个epoch}\_{第几个step}.ckpt。

查看损失函数随着训练步数的变化情况

steps = step\_loss["step"]

loss\_value = step\_loss["loss\_value"]

steps = list(map(int, steps))

loss\_value = list(map(float, loss\_value))

plt.plot(steps, loss\_value, color="red")

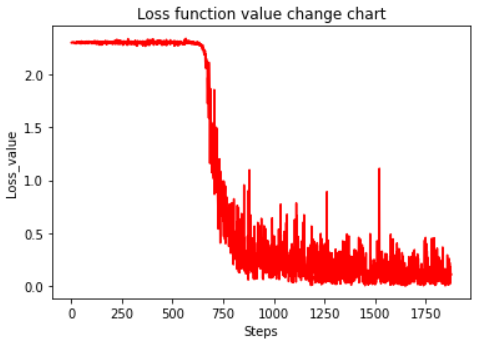
plt.xlabel("Steps")

plt.ylabel("Loss\_value")

plt.title("Loss function value change chart")

plt.show()

输出：



数据测试验证模型精度

搭建测试网络的过程主要为：

1、载入模型.cptk文件中的参数param；

2、将参数param载入到神经网络LeNet5中；

3、载入测试数据集；

4、调用函数model.eval()传入参数测试数据集ds\_eval，就生成模型checkpoint\_lenet-1\_1875.ckpt的精度值。

def test\_net(network, model, mnist\_path):

"""Define the evaluation method."""

print("============== Starting Testing ==============")

# load the saved model for evaluation

param\_dict = load\_checkpoint("checkpoint\_lenet-1\_1875.ckpt")

# load parameter to the network

load\_param\_into\_net(network, param\_dict)

# load testing dataset

ds\_eval = create\_dataset(os.path.join(mnist\_path, "test"))

acc = model.eval(ds\_eval, dataset\_sink\_mode=False)

print("============== Accuracy:{} ==============".format(acc))

test\_net(network, model, mnist\_path)

输出：

============== Starting Testing ==============

============== Accuracy:{'Accuracy': 0.969551282051282} ==============

经过1875步训练后生成的模型精度超过95%，模型优良。 我们可以看一下模型随着训练步数变化，精度随之变化的情况。acc\_model\_info()函数是将每125步的保存的模型，调用model.eval()函数将测试出的精度返回到步数列表和精度列表，如下：

def acc\_model\_info(network, model, mnist\_path, model\_numbers):

"""Define the plot info method"""

step\_list = []

acc\_list = []

for i in range(1, model\_numbers+1):

# load the saved model for evaluation

param\_dict = load\_checkpoint("checkpoint\_lenet-1\_{}.ckpt".format(str(i\*125)))

# load parameter to the network

load\_param\_into\_net(network, param\_dict)

# load testing dataset

ds\_eval = create\_dataset(os.path.join(mnist\_path, "test"))

acc = model.eval(ds\_eval, dataset\_sink\_mode=False)

acc\_list.append(acc['Accuracy'])

step\_list.append(i\*125)

return step\_list,acc\_list

# Draw line chart according to training steps and model accuracy

l1,l2 = acc\_model\_info(network, model, mnist\_path, 15)

plt.xlabel("Model of Steps")

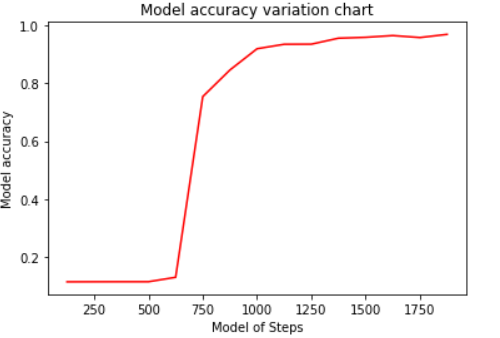
plt.ylabel("Model accuracy")

plt.title("Model accuracy variation chart")

plt.plot(l1, l2, 'red')

plt.show()

输出：



模型预测应用

使用生成的模型应用到分类预测单个或者单组图片数据上，具体步骤如下：

1、需要将要测试的数据转换成适应LeNet5的数据类型。

2、提取出image的数据。

3、使用函数model.predict()预测image对应的数字。需要说明的是predict返回的是image对应0-9的概率值。

4、调用plot\_pie()将预测的各数字的概率显示出来。负概率的数字会被去掉。

ds\_test = create\_dataset(test\_data\_path).create\_dict\_iterator()

data = ds\_test.\_\_next\_\_()

images = data["image"].asnumpy()

labels = data["label"].asnumpy() # The subscript of data picture is the standard for us to judge whether it is correct or not

output =model.predict(Tensor(data['image']))

# The predict function returns the probability of 0-9 numbers corresponding to each picture

prb = output.asnumpy()

pred = np.argmax(output.asnumpy(), axis=1)

err\_num = []

index = 1

for i in range(len(labels)):

plt.subplot(4, 8, i+1)

color = 'blue' if pred[i] == labels[i] else 'red'

plt.title("pre:{}".format(pred[i]), color=color)

plt.imshow(np.squeeze(images[i]))

plt.axis("off")

if color == 'red':

index = 0

# Print out the wrong data identified by the current group

print("Row {}, column {} is incorrectly identified as {}, the correct value should be {}".format(int(i/8)+1, i%8+1, pred[i], labels[i]), '\n')

if index:

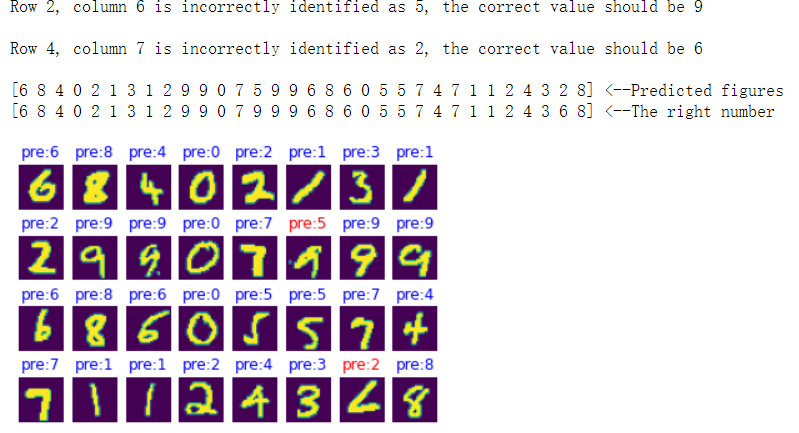
print("All the figures in this group are predicted correctly！")

print(pred, "<--Predicted figures") # Print the numbers recognized by each group of pictures

print(labels, "<--The right number") # Print the subscript corresponding to each group of pictures

plt.show()

输出：



构建一个概率分析的饼图函数。

# define the pie drawing function of probability analysis

def plot\_pie(prbs):

dict1 = {}

# Remove the negative number and build the dictionary dict1. The key is the number and the value is the probability value

for i in range(10):

if prbs[i] > 0:

dict1[str(i)] = prbs[i]

label\_list = dict1.keys() # Label of each part

size = dict1.values() # Size of each part

colors = ["red", "green", "pink", "blue", "purple", "orange", "gray"] # Building a round cake pigment Library

color = colors[: len(size)]# Color of each part

plt.pie(size, colors=color, labels=label\_list, labeldistance=1.1, autopct="%1.1f%%", shadow=False, startangle=90, pctdistance=0.6)

plt.axis("equal") # Set the scale size of x-axis and y-axis to be equal

plt.legend()

plt.title("Image classification")

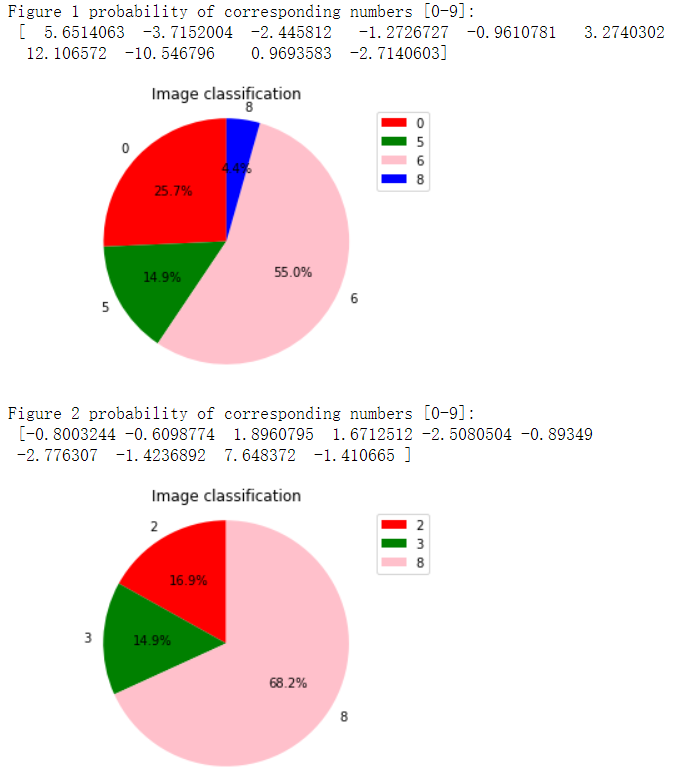
plt.show()

for i in range(2):

print("Figure {} probability of corresponding numbers [0-9]:\n".format(i+1), prb[i])

plot\_pie(prb[i])

输出：



## 实验总结

本章节主要通过华为深度学习框架MindSpore完成手写数字识别的全流程。