
Python与机器学习

Python篇

Python篇

第1讲

纲要

□ Python安装

□ Python基础

□ K近邻算法

Python安装

□ Python

- 作者： Guido Van Rossum
- 目前在Google工作

Rossum的段子：

- 去Google面试时，面试官问他做过什么
- “I wrote Python”



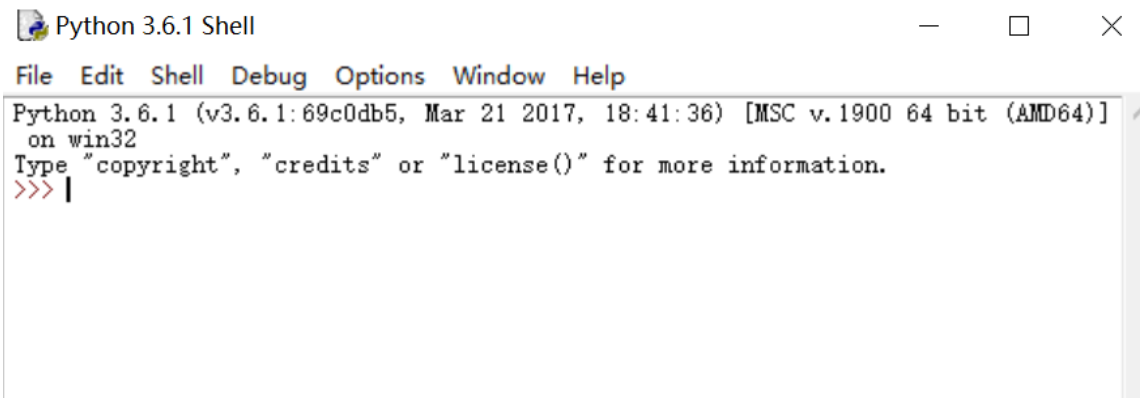
Python安装

□ www.python.org/downloads/

- Windows
- Mac OS
- Linux

□ 本课，我们用Windows最新版Python3.6.1

□ Python的开发环境有许多，我们采用Python自带的IDLE启动Python Shell(类似于命令行)



```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:41:36) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

纲要

□ Python安装

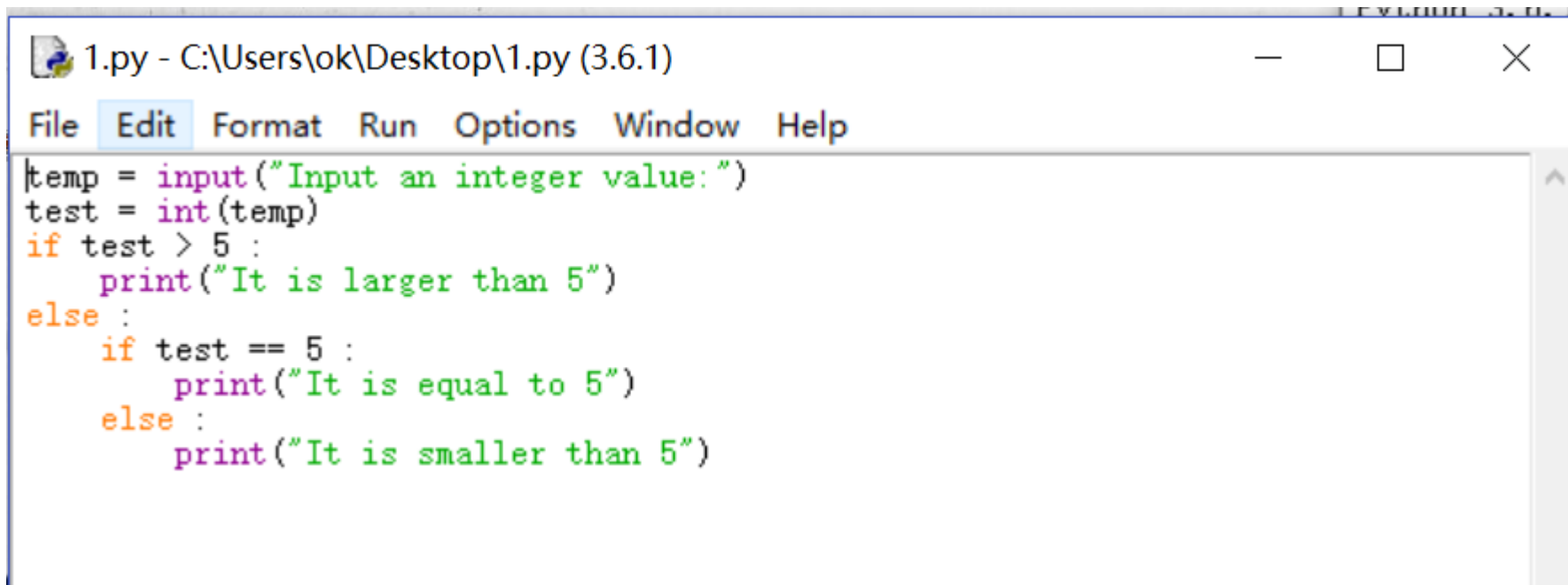
□ Python基础

□ k近邻算法

Python基础

□ 不要括号

- Python使用缩进而不是括号来进行代码段标识，减少了视觉上的混乱，并且使程序变短，提高了程序的可读性。



```
1.py - C:\Users\ok\Desktop\1.py (3.6.1)
File Edit Format Run Options Window Help
temp = input("Input an integer value:")
test = int(temp)
if test > 5 :
    print("It is larger than 5")
else :
    if test == 5 :
        print("It is equal to 5")
    else :
        print("It is smaller than 5")
```

Python基础

□ 变量

- 与其他语言不同之处，Python无需声明变量类型，直接赋值使用即可。
- 使用变量之前，必须先赋值
- 变量名：字母、数字、下划线；但不能以数字开头
- 字母区分大小写

□ 字符串

- 用单引号 ('), 或者双引号(")括起来
- 如果字符串要出现单引号或者双引号以及反斜杠等特殊字符
 - 可以用\ (path = 'C:\\python')
 - 或者前面添加r (path = r'C:\python')
- 长字符串可以用三引号表示
 - ''' 长字符串 '''

Python基础

□ 比较语句

- > >= < <= == !=

□ 条件语句

```
if 条件:  
    执行操作  
else:  
    执行操作
```

```
while 条件:  
    条件为真, 执行操作
```


□ 几个条件结合, 用 and

```
条件1 and 条件2
```

Python基础

□ 产生随机数

- 引入random
- import random
- random.randint(a,b) //产生[a,b]之间的整数
- random.random() //产生[0,1]之间的小数

 2.py - C:\Users\ok\Desktop\2.py (3.6.1)

File Edit Format Run Options Window Help

```
import random
random_integer = random.randint(1,10)
print(random_integer)
random_float = random.random()
print(random_float)
```

Python基础

□ 数值类型

- 整型
- 浮点型
- 布尔型
- e记法（科学记数法）

□ 类型转换

- `int()` // 转换成整型
- `float()` // 转换成浮点型
- `str()` // 转换成字符串

□ 获得类型的方法

- `type()` // 返回类型
- `isinstance(a,b)` // 判断参数a是否是b类型

Python基础

□ 算术操作符

- + - * / % (取余)

- 除法操作

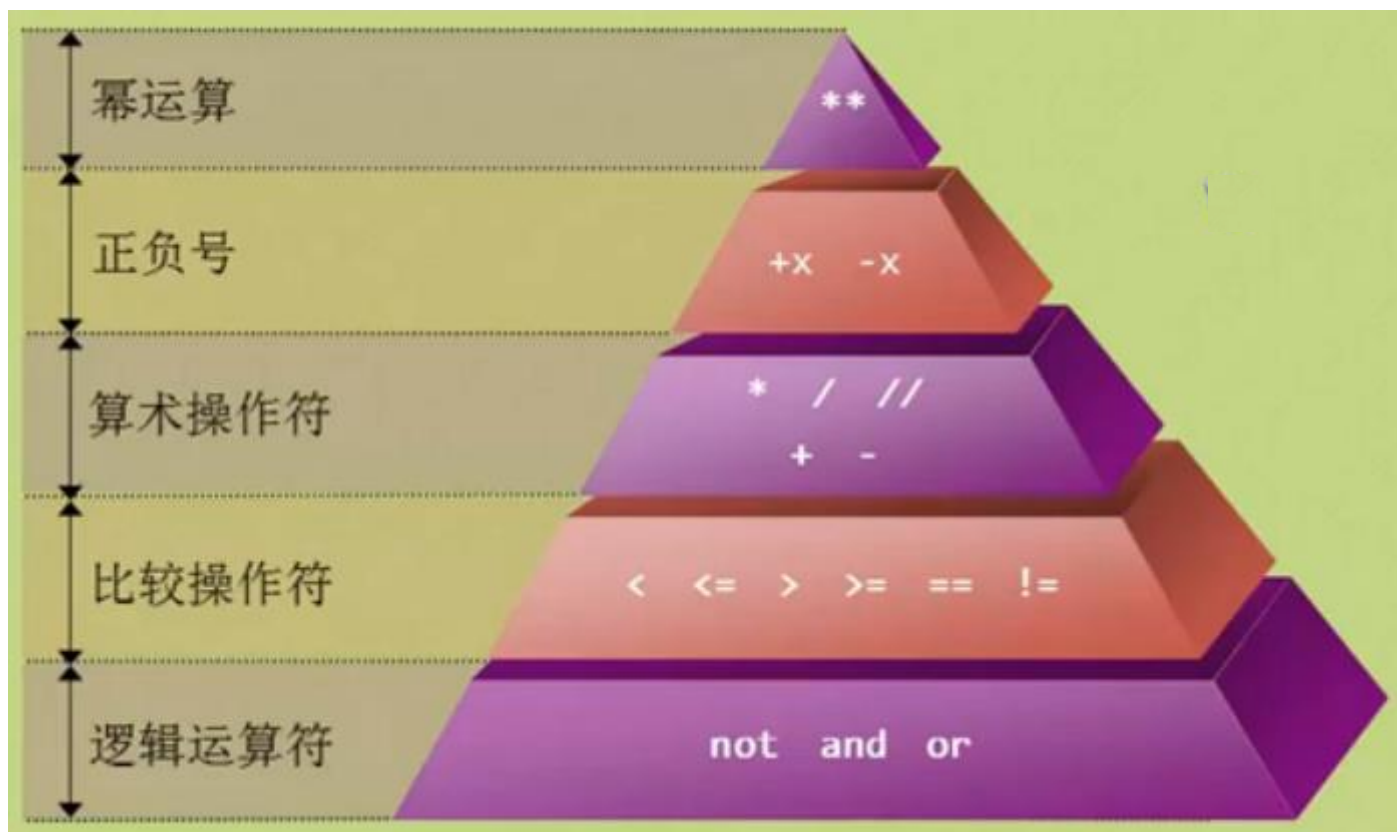
- /

- //

- 幂运算

- **

□ 优先级



Python基础

□ 三元操作符

```
x, y = 4, 5
if x < y :
    small = x
else:
    small = y
```

可写成：
small = x if x<y else y

Python基础

□ for循环

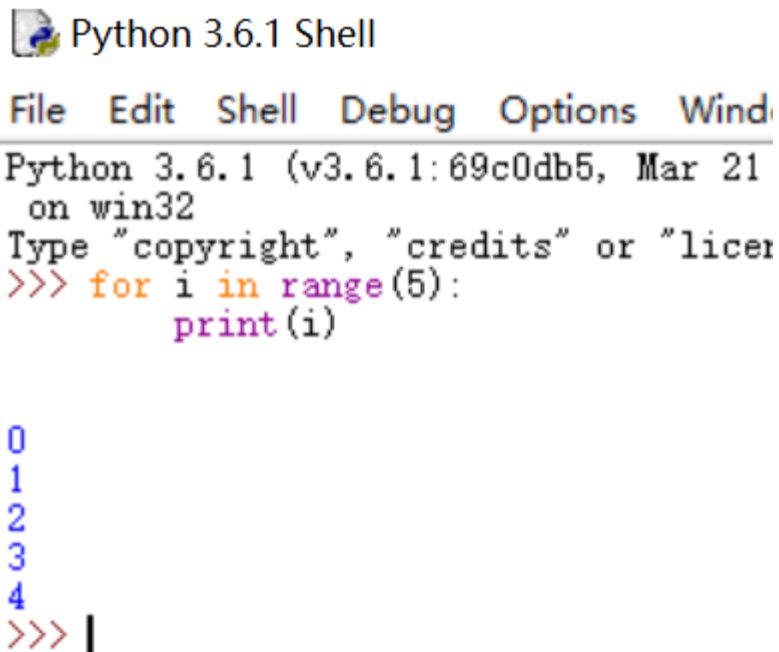
for 目标 in 表达式:
 循环体

□ range()函数

- range([start,] stop[, step=1])
- 作用：生成一个从start开始到stop
- range(1,5) // 表示[1, 2, 3, 4]
- range(1,5,2) // 表示[1,3], 从1开始到5，步长为2
- range(5) // 表示[0,1,2,3,4]

□ list()函数

- list(range(5)) // 输出[0,1,2,3,4]



```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017)
on win32
Type "copyright", "credits" or "license()" for more
>>> for i in range(5):
>>>     print(i)
0
1
2
3
4
>>> |
```

Python基础

□ break

- 跳出整个循环

□ continue

- 跳出本次循环

```
>>> for i in range(10):  
    if i == 5:  
        break  
    print(i)
```

输出:

```
0  
1  
2  
3  
4  
>>> |
```

```
>>> for i in range(10):  
    if i%2 != 0:  
        print(i)  
        continue  
    i += 2  
    print(i)  
|
```

输出:

```
2  
1  
4  
3  
6  
5  
8  
7  
10  
9
```

Python基础

□ 列表 list

```
score = ['A', 'B', 'C', 'D']
```

- 普通列表：保存一种类型元素
- 混合列表：保存多种类型元素

```
mix = ['A', 95, [70,25]]
```

- 空列表：empty_list = []

□ 向列表添加元素

- append() // score.append('NA') 在列表最后添加一个元素,
 // 一次只能添加一个元素
- extend() // score.extend(['NA', 'A+']) 在列表最后添加2个
- insert() // score.insert(0, 'A+') 在列表第0位添加元素A+

Python基础

□ 列表 list

```
score = ['A', 'B', 'C', 'D']
```

- 获取列表某个位置元素: `score[0]` // 输出'A'
- 删除元素
 - `score.remove('A')` // 删除列表score中的元素'A'
 - `del score[0]` // 删除列表score中的第0个元素
 - `del score` // 删除整个列表score
 - `score.pop()` // 返回列表score中最后一个元素, 并删除它
 - `score.pop(0)` // 返回列表score中的第0个元素, 并删除它
- 列表分片: 一次性获取多个元素 (列表分片)
 - `score[0:3]` // 返回列表score中第0个元素到第3-1个元素
 - `score[:3]` // 和`score[0,3]`等价
 - `score[3:]` // 返回列表score中第3个元素到最后一个元素
 - `score[:]` // 返回列表score的拷贝

```
Q: score2 = score[:]  
   score3 = score 有何区别 ?
```

Python基础

□ 列表 list 的常用操作符

● 列表的比较

```
score1 = [90]  
score2 = [85]  
score1 > score2 // 返回True
```

```
score1 = [90, 70]  
score2 = [85, 90]  
score1 > score2 // 返回True, 第一个元素的比较结果
```

● 列表拼接

```
score3 = score1 + score2 //返回[90, 70, 85, 90]  
score1 *= 2 // 返回[90, 70, 90, 70]并赋值给score1
```

● 列表元素查询

```
90 in score1 // 返回True  
90 not in score1 // 返回False
```

Python基础

□ 列表 list 的内置函数

● 通过dir(list)查询

```
>>> dir(list)
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__r
educe__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
'__setitem__', 'sizeof', 'str', 'subclasshook', 'append', 'clear',
'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort',
']
```

- score.count(90) // 返回列表score中90出现的次数
- score.index(90) // 返回列表score中90第一次出现的位置
- score.index(90,0,3) // 返回90在位置[0,3]中第一次出现的位置
- score.reverse() // 翻转列表，最后一个元素放到第一个...
- score.sort() // 对列表元素进行从小到大排序
- score.sort(reverse=True) // 对列表元素进行从大到小排序

Python基础

□ 元组 tuple

- 和列表类似，但是元组的元素不能修改
- 元组理论使用小括号（列表使用中括号）

```
>>> tuple1 = (1, 2, 3, 4, 5)
>>> type(tuple1)
<class 'tuple'>
>>> |
```

- 但是，元组也可以不要小括号

```
>>> tuple2 = 1, 2, 3, 4, 5
>>> type(tuple2)
<class 'tuple'>
>>> |
```

- 创建空元组： `tuple3 = ()`
- 元组只包含一个元素时： `tuple4 = (1,)` // 需要加逗号
- 元组的元素不允许修改和删除，但可以删除整个元组

Python基础（待补充）

□Python数组(三种类型):

- list (列表) tuple (元组)
- dictionary (字典)
 - 表达形式: (大括号)

```
score_dic = {'A' : 90, 'B' : 80}
```

- 与列表的区别: 字典是无序的, 通过键来访问成员

```
访问score_dic['A'], 得到90
```

- score_dic.clear() // 清空字典
- score_dic.keys() // 获得键的列表
- score_dic.values() // 获得值的列表
- score_dic.copy() // 复制字典
- score_dic.pop(k) // 删除键k
- score_dic.get(k) // 获得键k的值
- 可以通过help(dict)查看

Python基础

- ❑ 字典的排序：字典含有键和键值，无法对字典排序
- ❑ 但是，可以对键，或者键值排序
- ❑ `sorted(iterable, key, reverse)`函数，返回值类型为列表
 - `iterable`：可以迭代的对象，如`dict.items()`
 - `key`：一个函数，用来选取参与比较的元素
 - `reverse`：指定顺序还是倒叙排序， `True`=倒叙， `False`=顺序

- 按键排序：

```
>>> score_dic = {'B':80, 'A':90}
>>> sorted(score_dic.keys())
['A', 'B']
>>> sorted(score_dic.keys(), reverse=True)
['B', 'A']
```

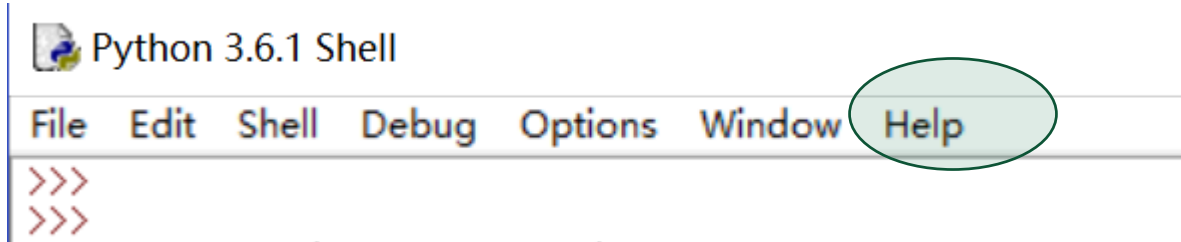
- 按键值排序：

```
>>> sorted(score_dic.items(), key=lambda item:item[1])
[('B', 80), ('A', 90)]
>>> sorted(score_dic.items(), key=lambda item:item[1], reverse=True)
[('A', 90), ('B', 80)]
```

- `# lambda x:x[1]` 表示一个函数，返回`x`的第1位值

Python基础

□ Python doc 或者 help(...)查看



Python基础

□ 函数

- 定义:

```
>>> def MyFirstFunction():  
    print('Hello world')
```

- 执行:

```
>>> MyFirstFunction()  
Hello world
```

- 定义（带参数）：

```
>>> def MySecondFunction(name):  
    print(name + ' Hello world')
```

- 执行:

```
>>> MySecondFunction('OK')  
OK Hello world
```

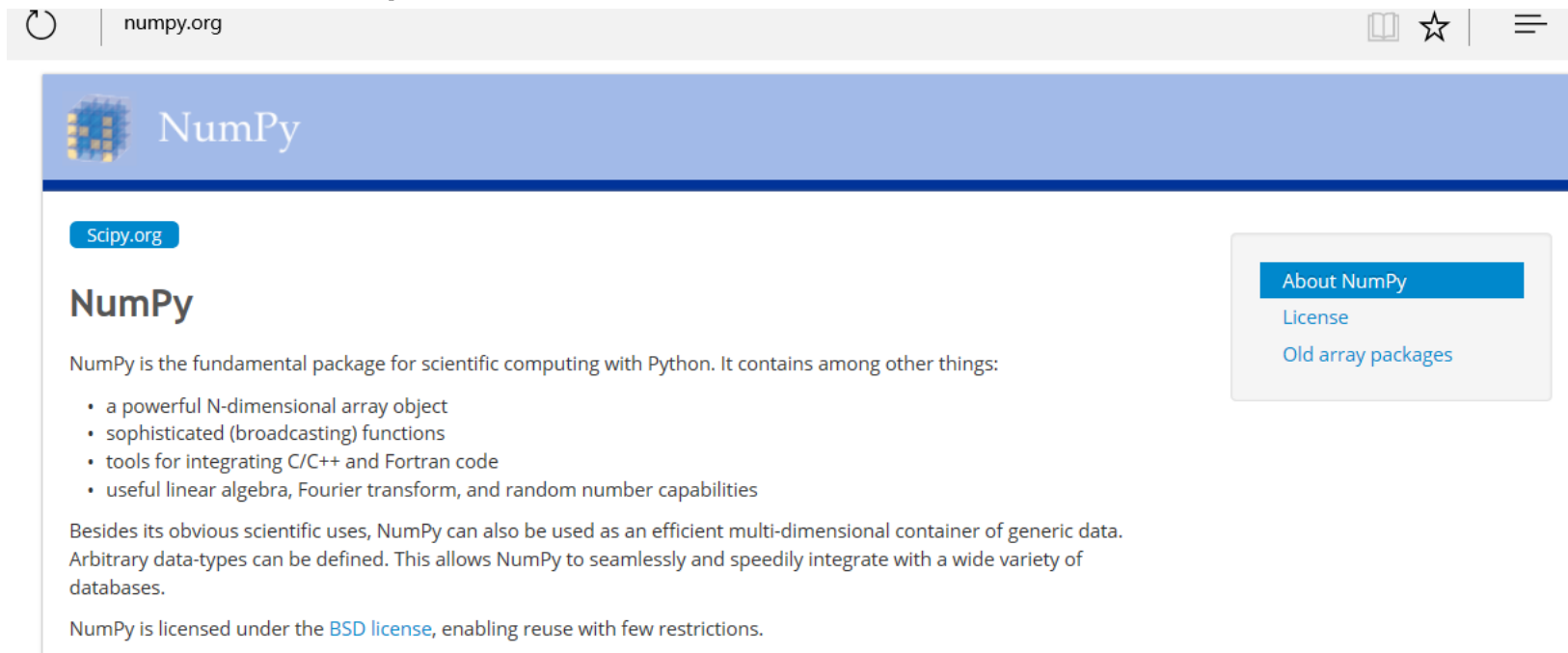
有返回的函数:

```
>>> def add(a, b):  
    return (a+b)
```


Python基础

□ 矩阵

● 安装NumPy模块（科学计算包）



● 在Python中实现免费的MatLab

Python基础

□ 安装NumPy包

- 要用到pip安装NumPy的whl包（提前将pip加入到系统变量中）
- <http://www.lfd.uci.edu/~gohlke/pythonlibs/> 下载NumPy
- cmd → pip install "Numpy.whl"

纲要

□ Python安装

□ Python基础

□ K近邻算法

k近邻算法

k近邻学习的工作机制

□ k近邻(k-Nearest Neighbor, kNN)学习是一种常用的监督学习方法：

- 确定训练样本，以及某种距离度量。
- 对于某个给定的测试样本，找到训练集中距离最近的k个样本，对于分类问题使用“投票法”获得预测结果，对于回归问题使用“平均法”获得预测结果。还可基于距离远近进行加权平均或加权投票，距离越近的样本权重越大。
 - 投票法：选择这k个样本中出现最多的类别标记作为预测结果。
 - 平均法：将这k个样本的实值输出标记的平均值作为预测结果。

k近邻算法

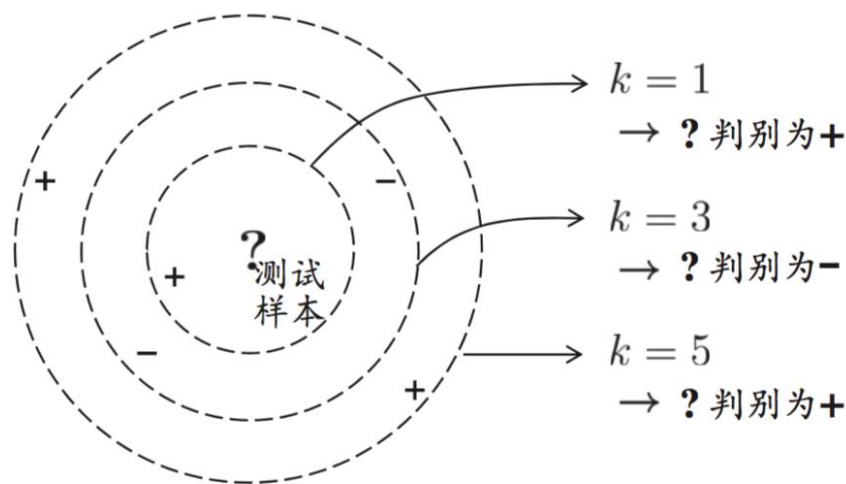


图 10.1 k 近邻分类器示意图. 虚线显示出等距线; 测试样本在 $k = 1$ 或 $k = 5$ 时被判别为正例, $k = 3$ 时被判别为反例.

□ k 近邻分类器中的 k 是一个重要参数, 当 k 取不同值时, 分类结果会有显著不同。另一方面, 若采用不同的距离计算方式, 则找出的“近邻”可能有显著差别, 从而也会导致分类结果有显著不同。

K近邻算法

□ 伪代码：

- Input: 测试样本, 训练样本, 训练样本的标签, k
- Output: 测试样本的标签
- 过程：
 - 1. 计算测试样本与每个训练样本的距离
 - 2. 按照距离进行排序
 - 3. 选取与测试样本距离最近的 k 个训练样本
 - 4. 投票法(计算这 k 个训练样本中每个标签出现的频率)
 - 5. 返回出现频率最高的标签, 即为测试样本的预测标签

K近邻算法

Python代码：

```
from numpy import * # 引入科学计算包
```

```
# 创建数据集
```

```
def createDataSet():
```

```
    group = array([1.0, 1.1], [1.0, 1.0], [0, 0], [0, 0.1])
```

```
    labels = ['A', 'A', 'B', 'B']
```

```
    return group, labels
```

```
>>> kNN.classify0([0.1, 0.2], group, labels, 3)
```

```
'B'
```

```
# k近邻算法
```

```
def classify0:
```

```
>>> import kNN
```

```
>>> group, labels = 1
```

```
>>> group
```

```
array([[ 1. ,  1.1],
       [ 1. ,  1. ],
       [ 0. ,  0. ],
       [ 0. ,  0.1]])
```

```
>>> labels
```

```
['A', 'A', 'B', 'B']
```

```
>>> |
```

```
|
```

```
|
```

```
f
```

```
    voteLabel = labels[sortedDistIndicies[i]]
```

```
    classCount[voteLabel] = classCount.get(voteLabel,0) + 1
```

```
# +1操作
```

```
# 排序 (对字典的键值进行排序)
```

```
sortedClassCount=sorted(classCount.items(),key=lambda itemLabel:itemLabel[1],reverse=True)
```

```
return sortedClassCount[0][0]
```

```
# 获取第一个键，即被预测的测试样本标签
```

w上复制x1-1次
样本的差值


和
每的矩阵
置结果（非矩阵值）

K近邻算法

- 若训练数据由文件保存：
 - Python中文件的读写操作

- 实例：

- 约会网站的推荐问题
 - 属性(形容一个人的特征):每年的飞行里程数/玩游戏时间/冰淇淋消耗量
 - 类别标签：不喜欢的人/魅力一般的人/极具魅力的人
- 收集了1000组训练样本数据，存于文件中
- Target：给定一个人物的属性，判断是否推荐该人(属于哪一类)

 datingTestSet2.txt - 记事本

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)
40920	8.326976		0.953952	3
14488	7.153469		1.673904	2
26052	1.441871		0.805124	1
75136	13.147394		0.428964	1

K近邻算法

□ 从文件中读取数据

从文件中读取数据到numpy矩阵中

```
def file2matrix(filename):  
    fr = open(filename, 'r')          # 打开(读)文件  
    linesInFile = fr.readlines()      # 读取整个文件，并按行以列表形式保存  
    numberOfLines = len(linesInFile)  # 返回文件中的行数(训练样本数)  
    returnMat = zeros([numberOfLines, 3]) # 准备一个numberOfLines * 3 的零矩阵  
    classLabelVector = []             # 准备一个标签列表  
    index = 0  
    for line in linesInFile:  
        line = line.strip()           # 删除字符串中头、尾出现的空白符（包括'\n', '\r', '\t', ' '）  
        listFromLine = line.split('\t') # 将字符串按'\t'进行分割，并返回一个列表  
        returnMat[index, :] = listFromLine[0:3] # 该列表前3位保存  
        classLabelVector.append(int(listFromLine[-1])) # 该列表最后一位保存(以int形式)  
        index += 1  
    fr.close()  
    return returnMat, classLabelVector
```

- 这样，我们就从文本文件中导入了数据，并以矩阵/列表的形式保存

```
datingDataMat, datingLabels = kNN.file2matrix('datingTestSet2.txt')
```

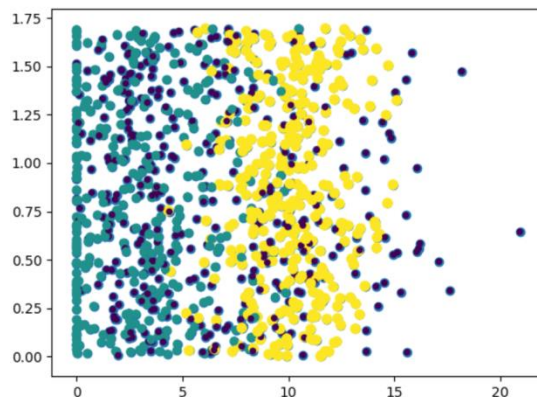
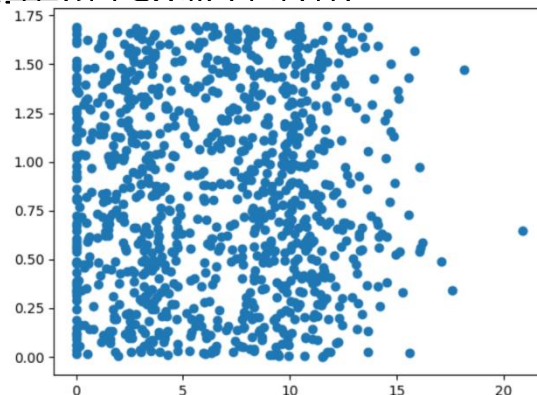
K近邻算法

□ Python画图：很多时候，我们想要图像显示数据特点

- Python的Matplotlib模块
 - 要用到pip安装Matplotlib的whl包
 - cmd → pip install "Matplotlib.whl"

□ 使用Matplotlib

```
>>> import matplotlib
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.scatter(datingDataMat[:,1], datingDataMat[:,2])
>>> plt.show()
```



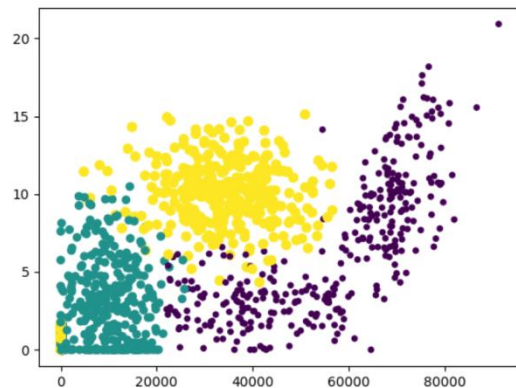
```
>>> ax.scatter(datingDataMat[:,1], datingDataMat[:,2], 15.0*array(datingLabels),
15.0*array(datingLabels))
```

K近邻算法

□ 使用Matplotlib

```
ax.scatter(datingDataMat[:,0], datingDataMat[:,1], 15.0*array(datingLabels),  
15.0*array(datingLabels))
```


用第0和第1列



K近邻算法

□ 归一化(Normalization)

● 训练数据类型

 datingTestSet2.txt - 记事本

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)
40920	8.326976		0.953952	3
14488	7.153469		1.673904	2
26052	1.441871		0.805124	1
75136	13.147394		0.428964	1

● 归一化处理

- $$\text{newValue} = (\text{oldValue} - \text{min}) / (\text{max} - \text{min})$$

```
def autoNorm(dataSet):  
    minVals = dataSet.min(0) # 按列返回最小值  
    maxVals = dataSet.max(0)  
    ranges = maxVals - minVals  
    normDataSet = zeros(shape(dataSet))  
    m = dataSet.shape[0] # 得到矩阵行数  
    normDataSet = dataSet - tile(minVals, (m,1))  
    normDataSet = normDataSet / tile(ranges, (m,1)) #element wise divide  
    return normDataSet, ranges, minVals
```

```
normMat, ranges, minVals = kNN.autoNorm(datingDataMat)
```

K近邻算法

□ 一个完整的kNN分类器解决约会网站的推荐问题

- 训练数据
- 测试数据

```
def datingClassTest():
    hoRatio = 0.01      # 测试样本比例
    datingDataMat, datingLabels = file2matrix('datingTestSet2.txt')    #load data set from file
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]      # 总样本数
    numTestVecs = int(m*hoRatio)    # 其中用来做测试样本的数目
    errorCount = 0.0
    for i in range(numTestVecs):    # 前numTestVecs个样本作为测试样本，其余为训练样本
        classifierResult = classify0(normMat[i,:], normMat[numTestVecs:m,:], datingLabels[numTestVecs:m], 3)
        print ("the classifier came back with: %d, the real answer is: %d" % (classifierResult, datingLabels[i]))
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print ("the total error rate is: %f" % (errorCount/float(numTestVecs)))
    print ("the error count is: %d" % (errorCount))
```

总结

□ Python基础

□ K近邻学习 --- 最简单最有效的分类方法

□ 用Python可以轻松写出一个kNN学习算法

- 1. 如何从文本中读入数据
- 2. 如何将数据转化为矩阵结构
- 3. 如何计算距离