# 机器学习算法笔记

# 参考：《机器学习实战》,廖雪峰Python2.7教程，吴恩达机器学习视频

## 基本概念

### 1.监督学习

1，回归（regression）：试着推测出一系列连续值属性，如根据面积区域估计房价

2，分类（classification）：试着推测出离散的输出值

### 2.无监督学习

主要用于分类，如聚类算法（clustering 哦），用于关联分析的Apriori算法，如何使用FP-Growth算法改进关联分析

### 3.非均衡分类问题

如吴恩达所说，在大多数情况下不同类别的分类代价并不相等。我们通常基于错误率俩衡量分类器任务的成功程度，实际上，这样的度量错误掩盖了样例如何被分错的事实。利用混淆矩阵（confusion matrix）工具可以帮助人们更好地了解分类中的错误。P128

对于一个二类问题的混淆矩阵，有如下概率：

正确率（Precision）= 预测为正例的样本中的真正正例的比例，又称 查准率

召回率（Recall）= 真实正例中被正确预测为正例的比例，又称 查全率

假设我们的算法输出结果在0-1之间，我们使用阈值0.5来预测真和假。如果我们希望查准率更高，那么可以使用比0.5更大的阈值；若追求查全率，可以选择比0.5小的阈值。我们可以通过计算F1值来选择这个阈值，F1=2，选择使得F1值最高的阈值。

,另一个用于度量分类中的非均衡性的工具是ROC曲线，横坐标：假阳率，反例中被预测为正例的比例；纵坐标：真阳率，正例中被预测为正例的比例。左下角的点所对应的是将所有例子判断为反例，右上角的点对应的是将所有例子判断为正例。

ROC曲线不但可以用于比较分类器，还可以基于成本效益（cost-versus-benefit）分析来做出决策。

在理想的情况下，最佳的分类器应该尽可能地出于左上角，这就意味着分类器在假阳率很低的同时获得了很高的真阳率。例如在垃圾邮件过滤中，这就相当于过滤了所有的垃圾邮件，但没有将任何合法邮件误识。

对不同的ROC曲线进行比较的一个指标是曲线下的面积（Area Under the Curve,AUC）。AUC给出的是分类器的平均性能值。一个完美分类器的AUC为1.0，而随机猜测的AUC则为0.5。

ROC曲线的绘制及AUC计算函数

见 

**基于代价函数的分类器决策控制**P131：除了调节分类器的阈值之外，我们还有一些其他可以用于处理非均衡分类的代价的方法。其中的一种称为代价敏感的学习（cost-sensitive learning）。

在分类算法中，我们有很多的方法可以用来引入代价信息。在AdaBoost中，可以基于代价函数来调整错误权重向量D。在朴素贝叶斯中，可以选择具有最小期望代价而不是最大概率的类别作为最后的结果…

**处理非均衡问题的数据抽样方法**：通过欠抽样（undersampling）或过抽样（oversampling）来调节数据集中的正例和反例数目。P132 另外一种更好的非均衡问题的处理方法，就是在训练分类器时将错误的代价考虑在内。

### 4. 数据规约，数据标准化，数据规范化

#### 1 数据规约

假设根据业务需求，从数据仓库中获取了分析所需要的数据，这个数据集可能非常大，而在海量数据上进行数据分析和数据挖掘成本很高。数据规约技术可以用来得到数据集的规约表示，使得数据集变小，但同时仍然近于保持原数据的完整性。也就是说，在规约后的数据集上进行挖掘，依然能够得到与使用原数据集近乎相同的分析结果。

​数据规约包括维归约，数量规约和数据压缩。

维归约用于​减少所考虑的随机变量或者属性的个数，方法包括小波变换和主成分分析，将原始数据投影到较小的空间。属性子集选择是一种规约方法，其中不相关，弱相关，或者冗余的属性、维会被删除。（所以这个就是特征选择、降维的过程）。

数量归约用较小的数据集替换原数据集。可选的方法包括利用直方图来近似数据的分布；对数据进行聚类，用聚类的簇代表替换实际数据；对数据进行抽样，包括簇抽样（数据被分为不相交的簇，对每个簇进行抽样），分成抽样，采用抽样进行数据规约的一个优点是，得到样本的花费正比例与样本的大小，而不是原数据集的大小。

数据压缩使用变换，得到原数据的规约或”压缩 “表示。如果利用压缩后的数据能够对原数据进行重构，而不损失信息，则该数据规约称为无损的。如果只能得到近似重构的原数据，则该数据规约称为有损的。维归约和数量规约也可以视为某种形式的数据压缩。​

#### 2 数据标准化（data standardization）和数据规范化（data normalization）

数据预处理概念中，术语“标准化”和“规范化”是可以互换使用的。

​所用的度量单位可能影响数据分析，例如把身高的度量单位从米变成厘米，重量单位由公斤变成磅，可能导致完全不一样的结果。为了避免对度量单位选择的依赖性，数据规范化试图赋予所有属性相同的权重，常用的方法包括三种，最小-最大规范化，z分数规范化。

最小-最大规范化，属性的最大、最小值分别为max，min，对某条样本的该属性v进行规范化时，新的​值=(v-min)/(max-min)。

z分数规范化（z-score），属性的均值为u，标准差为σ​，对某条样本的该属性v进行规范化时，新的值=（v-u）/σ。当属性的实际最大值、最小值未知，或者离群点左右了最小-最大规范化时，这种方法比较有效，即对离群点具有一定鲁棒性。

### 5. 模型融合方法

模型融合是kaggle等比赛中经常使用到的一个利器，它通常可以在各种不同的机器学习任务中使结果获得提升。顾名思义，模型融合就是综合考虑不同模型的情况，并将它们的结果融合到一起。模型融合主要通过几部分来实现：从提交结果文件中融合、stacking和blending。

**从提交结果文件中融合**

最简单便捷的方式就是从竞赛的提交结果文件中进行融合，因为这样做并不需要重新训练模型，只需要把不同模型的测试结果弄出来，然后采取某种措施得出一个最终结果就ok。

**多数表决融合**

首先证明一下为什么**模型融合能提高准确率**以及**对低相关的结果进行融合可以获得更好结果**。

用一个概率的例子来说明。假如现在有10条记录，每条记录能被正确分类的概率为70%，或者某个模型对这10条记录进行分类能获得70%的准确率。现在拟合三个相当的模型，采用多数表决的情况下，对每条记录，三个模型都判断正确的概率为0.7\*0.7\*0.7~=0.34，两个模型判断正确的概率为0.7\*0.7\*0.3\*3~=0.44，那么通过三个准确率0.7的模型来融合的话，理论上最终每条记录能被正确分类的概率提升到**0.78**！

周志华教授在他的著作《机器学习》提到，结果的差异性越高，最终模型融合出来的结果也会越好。同样用一个简单的例子来证明：接着上面的话题，假设现在三个模型预测出来的结果是

model1：1111111100 = 80% 准确率

model2：1111111100 = 80% 准确率

model3：1011111100 = 70% 准确率

如果把这三个模型结果用多数表决组合起来，那么最终结果是：1111111100 = 80%，这个结果跟第一、二个模型是一致的，也就是，这样的模型融合对最终结果**没有任何的提升**。

假如我们现在把三个模型结果改为：

model1：1111111100 = 80% 准确率

model2：0111011101 = 70% 准确率

model3：1000101111 = 60% 准确率

显然这三个模型之间的差异更大，而且表面来看性能也不如前面提到的三个模型，但它们融合出来的结果是：1111111101 =**90% 准确率**！

**加权表决融合**

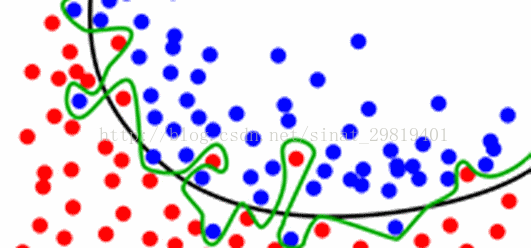
多数表决的融合方式默认了所有模型的重要度是一样的，但通常情况下我们会更重视表现较好的模型而需要赋予更大的权值。在加权表决的情况下，表现较差的模型只能通过与其他模型获得一样的结果来增强自己的说服力。

**对结果取平均**

对结果取平均在很多机器学习问题上以及不同的评估准则上都获得很不错的结果。

取均值的做法常常可以减少过拟合现象。在机器学习的应用上，过拟合现象是很普遍的，根本问题是训练数据量不足以支撑复杂的模型，导致模型学习到数据集上的噪音，这样产生的问题是模型很难泛化，因为模型“考虑”得过分片面。

但如果对结果取平均，可以在一定程度上减轻过拟合现象。图中所示，单个模型因为过拟合产生了绿色的决策边界，但事实上黑色的决策边界因为有更好的泛化能力从而有更好的效果。如果通过拟合多个模型并对模型结果取平均，对这些噪音点的考虑就会因为结果拉平均的原因而减少，决策边界也会慢慢的往黑色线靠拢。



记住，机器学习的目的并不是让模型记住训练数据，而是对未知数据有更好的推广。

**Stacking&Blending**

**Stacking**

**Stacking的基本思想是用一些基分类器进行分类，然后使用令一个分类器对结果进行整合。**

用2-fold stacking作为例子：

1.将训练数据分成A和B两份

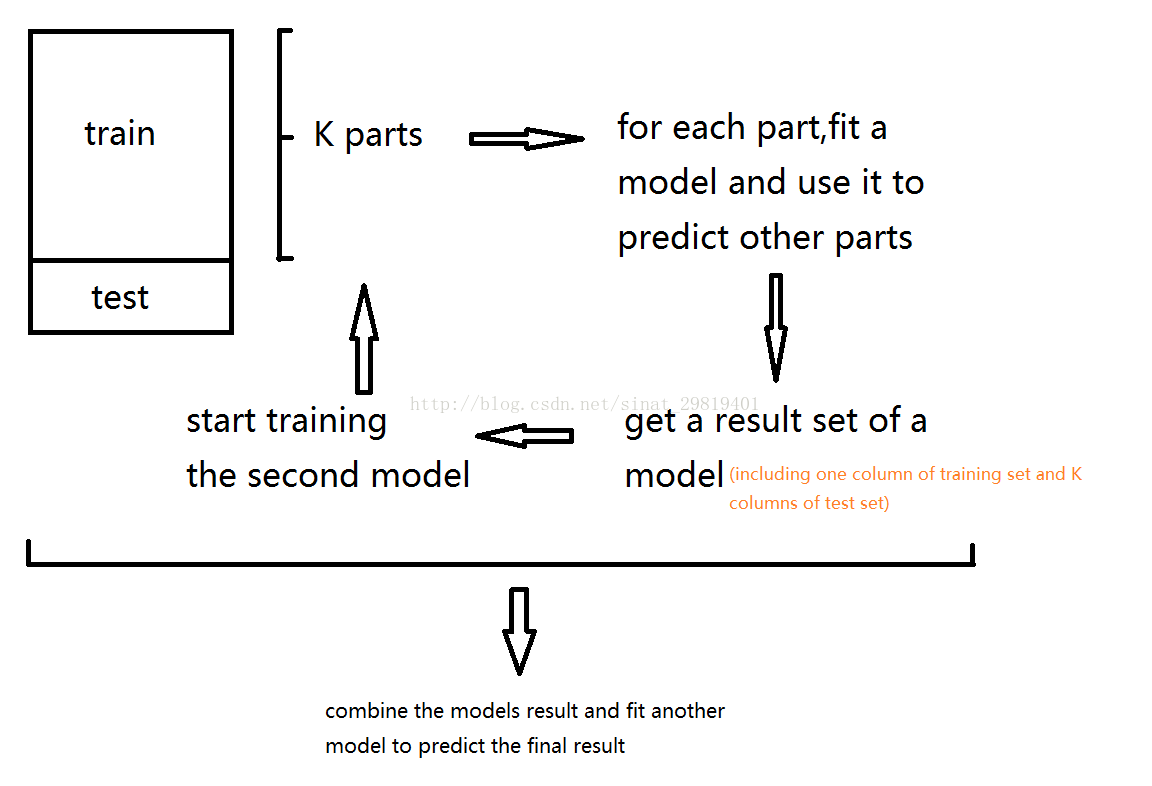
2.使用第一阶段模型用A训练，然后对B生成预测值

3.通过同样的模型用B训练，生成A的预测值

4.然后使用整个训练集来拟合这个模型，并生成测试集的预测值

5.像第（2）步一样训练第二阶段模型

Stacking的模型可以在特征空间上获取更加多的信息，因为**第二阶段模型是以第一阶段模型的预测值会作为特征**。



**Blending**

Blending与Stacking大致相同，只是**Blending的主要区别在于训练集不是通过K-Fold的CV策略来获得预测值从而生成第二阶段模型的特征，而是建立一个Holdout集**，例如说10%的训练数据，第二阶段的stacker模型就基于第一阶段模型对这10%训练数据的预测值进行拟合。说白了，就是把Stacking流程中的K-Fold CV 改成 HoldOut CV。

Blending的优点在于：

1.比stacking简单（因为不用进行k次的交叉验证来获得stacker feature）

2.避开了一个信息泄露问题：generlizers和stacker使用了不一样的数据集

3.在团队建模过程中，不需要给队友分享自己的随机种子

而缺点在于：

1.使用了很少的数据（第二阶段的blender只使用training set10%的量）

2.blender可能会过拟合（其实大概率是第一点导致的）

3.stacking使用多次的CV会比较稳健

对于实践中的结果而言，stacking和blending的效果是差不多的，所以使用哪种方法都没什么所谓，完全取决于个人爱好。

## Pycharm技巧

### 1 快捷方式

tab+0：光标恢复正常

ctrl+f：寻找代码中某单词

shift+tab：向左对齐一个tab

ctrl+/：注释

ctrl+alt+l :光标向左对齐

shift+enter：光标在中间时换行

ctrl+c 强行终止程序

若某行以\符号结束，意味着该行语句没有结束并会在下一行延续

### 2 常见bug解决（适用于Python 2.7）

# coding: utf-8 # 此行注释是告诉python解释器，按照UTF-8编码读取源代码

import sys

reload(sys)

sys.setdefaultencoding('utf8') # python的str默认是ascii编码，和unicode编码冲突,需要加上这几句

# from matplotlib.font\_manager import FontProperties

# font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14) # 设置作图中显示中文字体

Anaconda prompt 无法启动问题 只需要在环境变量中添加C:\Windows\System32即可 ，添加环境变量直接百度一下就出来了

pip install D:\Downloads\xgboost-0.80-cp27-cp27m-win\_amd64.whl 安装whl文件

## Python语法（适用于Python 2.7）

### 1 基础语法

#### 1.基本操作

print'the quick brown fox','jumps over','the lazy dog'

# 逗号在输出时变为空格

print '100+200=',100+200

# name=raw\_input() #程序等待用户输入，存入变量name中

name='patrik'

print 'hello,',name

#十六进制用0x前缀，例如0xff00;浮点数例如1.23e9,1.2e-5,注意整数与浮点数在计算机内部的存储方式不同，

#整数运算永远是精确的（因为整数除法永远是整数，%为取余），浮点运算可能会有误差。  
# \ 是转义符，转义字符\可以转义很多字符，比如\n表示换行，\t表示制表符，字符\本身也要转义，所以\\表示的字符就是\如果字符串里面有很多字符都需要转义，就需要加很多\，"I'm ok" 包含的字符个数为6；"I\'m \"OK\"!" 对应的输出为I'm "OK"!

# Python还允许用r''表示''内部的字符串默认不转义

# print r'\\\\t\\\'的输出为\\\\t\\\

True and False # 布尔运算，注意首字母要大写,None代表空值

# python中 = 代表着赋值，同一个变量可以反复赋值，而且是不同类型的变量

# 这就是灵活的动态语言

# 一般用全大写表示常量，实际上这个'常量'依然是变量

# Unicode将所有语言都统一到一套编码里，这样就不会有乱码问题了

# 在计算机内存中，统一使用Unicode编码，当保存时就转换为UTF—8编码

# 字符串'xxx'虽然是ASCII编码，但也可以看作UTF-8编码，而u'中文'则只能是Unicode编码

u'abc'.encode('utf-8') # 转换为UTF-8编码

u'中文'.encode('utf-8')

# 英文字符转换后表示的UTF-8的值和Unicode值相等（但占用的存储空间不同），

# 中文字符转换后1个Unicode字符将变为3个UTF-8字符，你看到的\xe4就是其中一个字节，

# 因为它的值是228，没有对应的字母可以显示，所以以十六进制显示字节的数值。

'abc'.decode('utf-8') #把UTF-8编码的字符串'xxx'转换为Unicode字符串

'\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')

#!/usr/bin/env python

#此行注释是为了告诉Linux/OS X系统，这是一个Python可执行程序，Windows系统会忽略这个注释

# -\*- coding:utf-8 -\*- #此行注释是告诉python解释器，按照UTF-8编码读取源代码

#格式化字符串的输出利用 % 实现

print 'hello,%s'%'world' # hello,world

print 'hi,%s,you have $%d.'%('patrik',1000000) # 填多个变量时，加上括号即可

# %d整数，%f浮点数，%s字符串，%x十六进制整数

# 如果不确定应该用什么，%s永远起作用，它会把任何数据类型转换为字符串

print '%2d-%02d'%(3,1) #输出 ' 3-01'

print '%.2f'%3.1415926 #输出 '3.14'

print 'age:%s.gender:%s'%(25,True) # age:25.gender:True

print u'hi,%s'%u'patrik' #对于Unicode字符串，最好确保替换的字符串也是Unicode字符串

print 'growth rate:%d %%'%7 #输出为 growth rate:7 %

#注意，在Python 3.x版本中，把'xxx'和u'xxx'统一成Unicode编码，即写不写前缀u都是一样的

#而以字节形式表示的字符串则必须加上b前缀：b'xxx'。

[Python内置函数(50)——print](https://www.cnblogs.com/sesshoumaru/p/6052719.html)

print(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

用于对象打印输出。通过命名参数sep来确定多个输出对象的分隔符(默认' ')，通过命名参数end确定输出结果的结尾(默认'\n')，通过命名参数file确定往哪里输出(默认sys.stdout)，通过命名参数fiush确定输出是否使用缓存(默认False)。参数None时意味着使用默认值。

>>> print(1,2,3,sep = '+',end = '=?')

1+2+3=?

#### 2.四种数据结构

# 使用list和tuple

#

#list是一种有序的集合，记得索引是从0开始的，classmate[0]

classmate=['patrik','bob','jack'] #用中括号[]定义list

l1=len(classmate) #获取list元素的个数

#索引超出范围会报错，最后一个元素的索引是len(classmate)-1,也可以用classmate[-1]来表示

classmate.append('mary') #在末尾追加元素

classmate.insert(1,'david') #把元素插入到指定位置

classmate.pop() #删除list末尾的元素

classmate.pop(1) #删除指定位置的元素

classmate[2]='sarah' #把某个元素替换为别的元素，直接赋值

#list里面的元素的数据类型可以不同

L=['apple',123,True]

s=['python','java',['asp','php'],'scheme']

l2=len(s) #注意l2=4，s[2]又是一个list，'php'=s[2][1],空的list的长度为0

#tuple（元组）与list非常类似，但是tuple一旦初始化就不能修改

#因为tuple不可变，所以代码更安全 定义一个空的tuple：t=()

t1=(1) #注意没加逗号的话，此时定义的是一个变量t1=1

t2=(1,) #用小括号()定义tuple，只有一个元素1

t=('a','b',['A','B'])

t[2][0]='C' #tuple不可变，但tuple中包含一个可变的list，所以可以直接改变这个值

# 引用list和tuple中元素的方式：[]

# 使用dict和set

#

# dict被称为字典，在其他语言中也称为map，使用key-value键值存储，用{}表示，查找速度极快

d={'patrik':100,'bob':85,'mary':75}

print d['patrik'] #无论这个表有多大，查找速度都不会变慢

d['adam']=67

d['mary']=76 #更新成绩

# 要避免key不存在的错误，可以通过in判断key是否存在，也可以通过dict提供的get方法

zaima = 'thomas' in d #zaima=False，dict中没有'thomas'这个key

d.get('thomas',-1) #如果thomas不存在，返回-1

d.pop('bob') #删除一个key，注意，dict内部存放的顺序和key放入的顺序没有关系

#dict特点：占用内存大，查找和插入的速度极快，不会随key的增加而变慢，哈希算法(Hash)通过key计算位置。

#牢记:dict的key必须是不可变对象，如字符串、整数，而list是可变的，因此不能作为key，最常用的key是字符串

#错误举例：key=[1,] d[key]='a list' 此时会提示error，因为[1,]是一个list，是可变的，不能作为key

#set也是一组key的集合，但不存储value，set中的key不能重复，用set()表示

#set和dict的唯一区别仅在于没有存储对应的value，同样不可以放入可变对象，下面会解释

#要创建一个set，需要提供一个list作为输入集合

s=set([1,2,3])

s=set([1,1,2,2,3,3]) #重复元素在set中自动被过滤

s.add(4) #添加了元素4到set中，可以重复添加，但无效

#set可以看作是数学意义上的无序和无重复元素的集合

s1=set([1,2,3])

s2=set([2,3,4])

s3=s1&s2 #得到的s3应该是set([2,3])

#补充：str字符串是不变对象，而list是可变对象

b=['c','b','a']

b.sort() # sort对列表进行排序（默认为升序），结果覆盖原列表，b变为['a','b','c']，

c='abc'

c.replace('a','A') # replace替换字符，c变成了'Abc'

#注意c的操作成功并不意味着字符串是可变的，c是个变量，对'abc'这个字符串进行replace操作以后，得到另一个字符串'Abc',再赋值给c

#tuple虽然是不变对象，但必须保证里面元素不包含list，否则不能作为key

注意：如果需要将集合作为字典的key使用，则必须要 map(frozenset,Cl)，将集合装换为不可变的frozenset

if can.issubset(tid):

# issubset用于判断tid是否存在于集合can中

if not ssCnt.has\_key(can): ssCnt[can]=1

# has\_key用于判断键是否存在于字典中

#### 3.条件判断和循环

#

age=20

if age>=18: #python的缩进非常严格

print 'your age is',age

print 'adult'

elif age>=6:print 'teenager'

else:print 'kid'

for person in classmate: #把每个元素代入变量person

print person

for x in range(101): #range(16)生成0-100的整数序列

sum=sum+x

print sum

sum=0;n=99; #计算100以内的所有奇数

while n>0:

sum=sum+n

n=n-2

print sum

# 注意从raw\_input()读取的内容永远以字符串形式返回，所以要先用int()把字符串转换为我们想要的整型

# 如果输入的是字母，那么程序会报错

birth=int(raw\_input('birth:'))

if birth<2000:

print '00前'

else:

print '00后'

#### 4.函数

#

# 求绝对值的函数 abs ，可使用 help(abs)查看函数信息

# cmp(x,y) 若x<y,返回-1；x==y，返回0；x>y,返回1

# 下面介绍数据类型转换

int('123');int(12.34);float('12.34');str(1.23); # 第三个将字符串转浮点，最后一个转字符串

unicode(100);bool(1);bool('') #后二分别返回True与False

d=abs #变量d指向abs函数

d(-1) #结果为1

# python中，定义一个函数要用def语句，依次写出函数名、函数参数和冒号，注意要在缩进体中写函数，函数值用return返回

# 当传入了不恰当的参数时，内置函数abs会检查出参数错误，我们自己写的函数可以用 isinstance 实现数据类型检查

def my\_abs(x):

if not isinstance(x,(int,float)): # 如果x不是整型或浮点型

raise TypeError('bad opend type')

if x>=0: # 注意有三个冒号

return x

else: return -x # 若没有return语句，函数执行完后返回的结果为None

def nop():

pass # pass作为占位符，还没想好怎么写的函数，先放个pass，让代码能运行起来

age=19；

if age>=18:

pass

# 函数可以返回多个值(实际上是一个tuple)，比如在游戏中经常需要从一个点移动到另一个点，给出坐标、位移和角度，

# 就可以计算出新的坐标

import math

def move(x,y,step,angle=0):

nx=x+step\*math.cos(angle) # math包中的cos函数，调用方法：math.cos

ny=y-step\*math.sin(angle)

return nx,ny

x,y=move(100,100,60,math.pi/6);print x,y

r=move(100,100,60,math.pi/6);print r #后面输出的结果比前面多一对（）

# 设置默认参数

def power(x,n=2): #计算x的n次方

s=1

while n>0:

n=n-1

s=s\*x

return s

power(3,2);power(3) #这两个得到的结果是一样的

# 由于我们经常计算x的平方，所以可以将第二个参数n的默认值设为2,如果要改变阶次，必须写 power(3,4)

# 当函数有多个参数时，把变化大的参数放前面（必选参数），变化小的参数放后面（默认参数）

# 注意必选参数在前，默认参数在后，否则会报错

def enroll(name,gender,age=6,city='beijing')

print 'name:',name

print 'gender',gender

print 'age',age

print 'city',city

# 绝大多数都符合默认参数，少数不符合的才需要提供额外的信息

enroll('sarah','F')

enroll('bob','M',7)

enroll('adam','M',city='tianjin') # 当不按顺序提供部分默认参数时，需要把参数名（譬如city）也写上

def add\_end(L=[]): #传入一个list作为参数

L.append('END') #末尾加上END

return L

add\_end([1,2,3])

add\_end(['x','y','z']) #输出为['x','y','z','END'] 输出正常

add\_end() #输出为 ['END']

add\_end() #输出为['END','END']

# 由于每次调用add\_end(),L都被赋值为['END'],若没有给它另外赋值，他就保持为['END']

# 因此，定义默认参数要牢记一点：默认参数必须指向不变对象,例如str或None

def add\_end(L=None):

if L is None: # if L is None ，好直白的语法...当后面为逻辑类型的时候，直接用is就代替赋值

L=[] # 当输入 add\_end()，参数是空的，所以默认L=None

L.append('END')

return L

add\_end();add\_end() #无论调用多少遍都不会出现问题

# 为什么要设计str、None这样的不变对象呢？因为不变对象一旦创建，对象内部的数据就不能修改，

# 这样就减少了由于修改数据导致的错误。此外，由于对象不变，多任务环境下同时读取对象不需要加锁，可以同时读

# 在python函数中，还可以定义可变参数。可变参数指的是传入的参数个数是可变的，可以从0，1一直到无限个

# 以数学题为例子，给定一组数目a,b,c...（个数不定）,计算a^2+b^2+c^2...

# 由于参数个数不确定，首先想到可以把a,b,c...作为一个list或tuple传进来

def calc(numbers):

sum=0

for n in numbers:

sum=sum+n\*n

return sum

# 但是调用的时候，需要先组装一个list [1,2,3] 或tuple

print calc([1,2,3]) #输出为14

# 若将函数的参数改为可变参数,在参数前面加上\*即可！

def calc(\*numbers):

sum=0

for n in numbers:

sum=sum+n\*n

return sum

calc(1,2,3) #简化了调用函数的方式

calc() #参数个数为0也没关系

# 如果已经有一个list或者tuple，如何调用一个可变参数？

# python允许你在list或tuple前面加一个\*号，把list或tuple的元素变为可变参数传进来

nums=[1,2,3]

calc(\*nums) #相当于 calc(nums[0],num[1],num[2])

# 可变参数允许你传入0个或任意个参数，这些可变参数在函数调用时自动组装为一个tuple。

# 而关键字参数允许你传入0个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个dict

def person(name,age,\*\*kw): # kw为关键字参数

print 'name:',name,'age:',age,'other:',kw

# 函数person除了必选参数name和age外，还接受关键字参数kw。在调用该参数时，可以只传入必选参数

person('patrik',30) # 输出结果 name:patrik age:30 other:{}

# 也可以传入任意个数的关键字参数

person('bob',35,city='beijing') # 输出结果 name:bob age:35 other:{'city':'beijing'}

person('adam',45,gender='m',job='engineer')

# 输出结果 name:adam age:45 other:{'gender':'m','job':'engineer'}

# 如上所见，关键字参数的作用在于它可以扩展函数的功能。

# 比如，在person函数里，我们保证能接收到name和age这两个参数，但如果参与者愿意提供更多的参数，我们也能收到。

# 和可变参数类似，也可以先组装一个dict，然后将dict转换为关键字参数传进去

kw={'city':'beijing','job':'engineer'}

person('jack',24,\*\*kw) # 或者 person('jack',24,city=kw['city'],job=kw['job'])

# 在python中定义函数，可同时使用四种参数

# 但参数定义的顺序必须是：必选参数、默认参数、可变参数和关键字参数

def func(a,b,c=0,\*arge,\*\*kw):

print 'a=',a,'b=',b,'c=',c,'args=',args,'kw=',kw

# 在函数调用的时候，python解释器自动按照参数位置和参数名把对应的参数传进去

func(1,2)

# 输出结果 a=1 b=2 c=0 args=() kw={}

func(1,2,3,'a','b',x=99)

# 输出结果 a=1 b=2 c=3 args=('a','b') kw={'x':99}

# 最神奇的是通过一个tuple和dict，也可以调用该函数

args=(1,2,3,4)

kw={'x':99}

func(\*args,\*\*kw)

# 所以对于任意函数，都可以通过 func(\*args,\*\*kw) 的形式调用它

# 小结

# 默认参数一定要用不可变对象，如果是可变对象，运行会有逻辑错误

# 要注意定义可变参数和关键字参数的语法:

# \*args 是可变参数，args接收的是一个tuple

# \*\*kw 是关键字参数，kw接收的是一个dict

# 可变参数既可以直接传入：func(1,2,3)

# 又可以先组装list或tuple，再通过\*args传入：func(\*(1,2,3))

# 关键字参数既可以直接传入：func(a=1,b=2)

# 又可以先组装dict，再通过\*\*kw穿入：func(\*\*{'a':1,'b':2})

# 使用 \*args和\*\*kw是python的习惯写法，最好使用习惯用法

# 递归：函数在内部调用自身

# 计算阶乘 n!=1\*2\*3\*...\*n,用函数fact(n)表示，可以看出：

# fact(n)=n!=1\*2\*...\*(n-1)\*n=(n-1)!\*n=fact(n-1)\*n

def fact(n):

if n=1:

return 1

return n\*fact(n-1)

# 理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰

# 使用递归函数需要注意防止栈（stack）溢出，每当进入一个函数调用，栈就会加一层栈帧，

# 每当函数返回，栈就会减一层栈帧。当递归调用的次数过多时，会导致栈溢出

# 解决递归调用栈溢出的方法是通过 尾递归 优化，可以把循环看成是一种特殊的尾递归函数

# 尾递归是指，在函数返回的时候，调用自身本身，且return语句不能包含表达式(可以是函数形式)

def fact(n):

return fact\_iter(n,1)

def fact\_iter(num,product):

if num==1:

return product

return fact\_iter(num-1,num\*product)

# 尾递归调用时，如果做了优化，栈不会增长，因此能解决栈溢出的问题了

# 但是！！！大多数编程语言没有针对尾递归做优化，包括python解释器。。。所以暂时没什么用

### 2.高级特性

#### 1.切片

L=['patrik','sarah','tracy','bob','jack']

# 取前n个元素，即索引为0-(N-1)的元素，可以用循环

r=[]

n=3

for i in range(n): # range(n)=[0,1,2] 生成一个list

r.append(L[i])

print r

# 切片（slice）用于取指定索引范围

print L[0:3] # 取出前3个元素，不包括3，当索引为0，可以省去，L[:3]

print L[1:3] # 从索引1开始，取出2个元素，即第二个和第三个元素

print L[-2:] # 输出为 ['bob','jack'],输出索引为-2和-1的，即取出倒数第二个和倒数第一个

print L[-2:-1] # 输出为 ['bob']，注意倒数第一个元素的索引是-1

L=range(100) # 索引为0-99，共100个索引

L[:10] # 通过切片轻松取出前10个数，索引为0 —— 9

L[-10:] # 取出后10个数，索引为-10 —— -1

L[10:20] # 取出前11到20的数，索引为10——19

L[:10:2] # 前10个数，每两个取一个

L[::5] # 所有数，每5个取一个

# tuple也可以用切片操作，只是操作的结果依然是tuple

print (0,1,2,3,4,5,6)[:3] # 输出结果为 (0,1,2)

# 再比如 字符串

'ABCDEFG'[:3]

'ABCDEFG'[::2] # 输出结果为 'ACEG'

#### 2.迭代（遍历list等可迭代对象）

#

# 释义：通过for循环遍历某个给定的list或tuple

# python的for循环不仅可以用在list和tuple上，还可以作用在其他可迭代对象上，譬如dict

d={'a':1,'b':2,'c':3}

for key in d:

print key # 输出的结果可能是 a c b

# 因为dict的存储不是按照list的方式顺序排列的，所以迭代出的结果顺序很可能不一样

for value in d.itervalues(): # 迭代value，利用itervalues函数

print value

for key,value in d.iteritems(): # 同时迭代key和value，利用iteritems函数，注意格式，key在前

print key,':',value

for ch in 'ABC':

print ch

# 利用 collections 模块的Iterable类型 判断 一个对象是否为可迭代对象

from collections import Iterable

isinstance('abc',Iterable) # str是否可迭代 输出为 True

isinstance([1,2,3],Iterable) # list也可迭代

isinstance(123,Iterable) # 整数当然就不可迭代啦

# 如果要对list实现类似java那样的下标循环，可以使用python内置的enumerate函数，将list变成索引—元素对

for i,value in enumerate(['A','B','C']):

print i,value

# 输出为 0 A 1 B 2 C

for x,y in [(1,1),(2,4),(3,9)]:

print x,y

#### 3.列表生成式

#

#

# 即 List Comprehensions，是python内置的简单而强大的可以用来创建list的生成式

# 举个栗子，要生成[1\*1,2\*2...10\*10]这样一个列表，可以用如下的一个循环来做

range(1,11)

L=[]

for x in range(1,11):

L.append(x\*x)

# 上面的循环太繁琐，而列表生成式一句话就ok了

[x\*x for x in range(1,11)]

[x\*x for x in range(1,11) if x%2==0] #加入判断，筛选出仅偶数的平方

[m+n for m in 'ABC' for n in 'XYZ'] #两层循环

# 输出结果：'AX','AY','AZ','BX','BY'...'CZ'

# 运用列表生成式，可以写出非常简洁的代码，例如列出当前目录下的所有文件和目录名

import os #导入os模块

print [d for d in os.listdir('.')] # os.listdir可以列出文件和目录

# 列表生成式也可以使用两个变量来生成list，注意这个 + 的用法

d={'x':'A','y':'B','z':'C'}

[k + '=' + v for k,v in d.iteritems()] # output: ['y=B','x=A','z=C']

# 把一个list中所有的字符串变为小写，利用lower()函数

L=['HEllo','woRld','ibm','apPle']

[s.lower() for s in L] # 注意此处lower的用法

#### 4.生成器（把一个列表生成式的[]改为()或将print改为yield）

#

#

# 通过列表生成器，我们可以直接创建一个列表，创建一个包含100万个元素的列表，但这不仅占用很大的存储空间，

# 而且如果我们仅仅需要访问前面几个元素，那后面绝大多数元素占用的空间都白白浪费了。

# 如果列表元素可以按照某种算法推算出来，这样就不必创建完整的list了，从而节省大量的空间

# 在python中，这种一边循环一边计算的机制，称为生成器 generator

# 制作生成器的方法1：把一个列表生成式的[]改为()

L=[x\*x for x in range(10)] # L是list

g=(x\*x for x in range(10)) # g是generrator

# 如果要一个一个打印出来

print g.next

print g.next

print g.next

# generator 保存的是算法，每次调用next(),就计算出下一个元素的值，直到最后一个元素，抛出Stoplteration的错误

for n in g:

print n # 正确的输出姿态

# 下面用函数的方式生成斐波拉契数列 1 1 2 3 5 8 13 21 34

def fib(max):

n=0;a=0;b=1

while n<max:

print b, # print后不加逗号默认换行输出，加逗号变为空一格输出

a,b = b,a+b # 赋值运算，先计算右边 b=1，a+b=1 再赋值给左边，即a=1,b=1

n=n+1

# 要把fib函数变为generator，只需把 print b 改为 yield b

# 这就是定义generator的另一个方法，如果一个函数定义中包含yield关键字，那么这就是一个generator

# 不同点在于：变成generator的函数，在每次调用next()的时候执行，遇到yield语句就中断，

# 再次执行时从上次中断的yield语句处继续执行。

def odd():

print 'step 1'

yield 1

print 'step 2'

yield 3

print 'step 3'

yield 5

o = odd()

# 只有每次输入 o.next() 才会继续输出 ,第四次调用o.next()时，没有yield可执行，所以就会报错

### 3.函数式编程

# 把大段代码拆成函数，通过一层层的函数调用，就可以将复杂任务分解为简单任务，这种分解就就叫做面向过程的程序设计

# 纯函数：只要输入是确定的，输出就是确定的，称之为没有副作用

# 而允许使用变量的程序设计语言，由于函数内部的变量状态不确定，同样的输入可能得到不同的输出

# 函数式编程允许把函数本身作为参数传入另一个函数，还允许返回一个函数，python允许使用变量，因此它不是纯函数式编程语言

#### 1.返回函数

#

def calc\_sum(\*args): # 这是一个可变参数

ax=0

for n in args:

ax=ax+n

return ax # 实现了一个可变参数的求和

# 但是，如果不需要立刻求和，而是在后面的代码中根据需要再计算。可以返回求和的函数

def lazy\_sum(\*args):

def sum():

ax=0

for n in args: # 函数在其定义内部引用了局部变量args

ax=ax+n

return ax

return sum # 返回的是求和的函数

f=lazy\_sum(1,3,5,7,9) # 调用lazy\_sum()时，返回的是求和函数，即将求和函数赋值给了一个变量f

f() # 调用函数f时，才真正计算求和的结果

# 在函数lazy\_sum中定义了函数sum，sum可以引用lazy\_sum的参数和局部变量

# 当lazy\_sum返回函数sum时，相关参数和变量都保存在返回的函数中，这种程序结构被称为“闭包(closure)”

# 注意：当调用lazy\_sum()时，每次调用都会返回一个新的函数，即使传入的是相同的参数

def count():

fs=[]

for i in range(1,4):

def f():

return i\*i

fs.append(f)

return fs

f1,f2,f3 = count() # 调用count函数，注意！此时没有执行f函数指令!!!f1,f2,f3三个函数返回以后才会执行

# f1=count() i=1 f2...i=2 f3...i=3

f1();f2();f3() # 因此，调用函数的结果并不是1，4，9。但实际结果是9，9，9。全部都是9！

# 原因在于返回的函数引用了变量i，但它并非立刻执行。等到3个函数都返回时，它们所引用的变量i已经变成了3

# 返回闭包时牢记的一点：返回函数不要引用任何循环变量，或后续会发生变化的变量

# 若要引用循环变量，要再创建一个函数，用该函数的参数绑定循环变量当前的值

def count():

fs=[]

for i in range(1,4):

def f(j):

# 不懂，用该函数的参数绑定循环变量当前的值？怎么办到的？？？

def g():

return j\*j

return g

fs.append(f(i))

# 调用count函数时，这一句执行了，然后就用j绑定了此时i的值吗？？？？

return fs

f1,f2,f3 = count() # 缺点是代码较长，可利用匿名函数（lambda）缩短代码。

print f1(),f2(),f3()

#### 2.匿名函数lambda

# 当我们在传入函数时，有时不需要显式地定义函数，直接传入匿名函数更方便

map(lambda x:x\*x,[1,2,3,4,5,6,7,8,9]) # output: 2,4,9,16,25,36...81

# 匿名函数 lambda x:x\*x 实际上如下。关键字lambda表示匿名函数，冒号前面的表示函数参数，后面的是函数内容

# 匿名函数只能有一个表达式，不用写return，返回值就是该表达式的结果

f=lambda x:x\*x # 将匿名函数赋值给一个变量f

f(4) # 再利用变量来调用该函数

def build(x,y):

return lambda:x\*x+y\*y # 将匿名函数作为返回值返回

# print build(4,2)() 注意lambda也是一个函数，所以要采用左边所示的方法调用

# 注意，python对匿名函数的支持有限，只有一些简单的情况下可以使用匿名函数

#### 3.装饰器

def now():

print '2013-12-25'

f=now # 通过变量调用函数

f()

# 函数对象有一个\_\_name\_\_属性，可以得到函数的名字

print now.\_\_name\_\_ # output: now

print f.\_\_name\_\_ # output: now

# 现在，假设我们要增强now()函数的功能，但又不希望修改now()函数的定义

# 这种在代码运行期间动态增加功能的方式，就称之为“装饰器” decorator

def log(func):

def wrapper(\*args,\*\*kw):

print 'call %s():' % func.\_\_name\_\_ # 在函数调用前后自动打印日志

return func(\*args,\*\*kw)

return wrapper

# 上面的log就是一个decorator，它接受一个函数func作为参数，并返回一个函数wrapper

# 借助python的 @ 语法，把decorator置于函数的定义处

@log

def now():

print '2013-12-25'

now() # 调用now()函数，不仅会运行now()函数本身，还会在运行now()函数前打印一行日志

# output: call now():

# 2013-12-25

# 把 @log 放在 now()函数的定义处，相当于执行了语句： now=log(now)

# 原来的now()函数仍然存在，只是现在同名的now变量指向了新的函数，

# 于是调用now()将执行新函数，也就是在log()函数中返回的wrapper()函数。

# wrapper()函数的参数定义式(\*args,\*\*kw),因此，wrapper()函数可以接受任意函数的调用

# 在wrapper()函数内，首先打印日志，再紧接着调用原始函数

#

# 如果decorator本身需要传入参数，那就需要编写一个返回decorator的高阶函数，

# 没看懂，详细见此：

https://www.liaoxuefeng.com/wiki/001374738125095c955c1e6d8bb493182103fac9270762a000/001386819879946007bbf6ad052463ab18034f0254bf355000

# 一个完整的decorator的写法如下：（没看懂）

import functools # 导入functools模块,这个模块通常可以用来增加函数的功能

def log(func):

@functools.wraps(func) #只需记住在定义wrapper()的前面加上@functools.wraps(func)即可

def wrapper(\*args, \*\*kw):

print 'call %s():' % func.\_\_name\_\_

return func(\*args, \*\*kw)

return wrapper

# 或者针对带参数的decorator：

import functools

def log(text):

def decorator(func):

@functools.wraps(func)

def wrapper(\*args, \*\*kw):

print '%s %s():' % (text, func.\_\_name\_\_)

return func(\*args, \*\*kw)

return wrapper

return decorator

#### 4.偏函数

#

#

# python的functools模块提供了很多功能，其中一个就是偏函数（partial function）

int('12345') # 传入字符串时，int()函数默认按十进制转换

int('12345',base=8) # 按8进制转换

int('12345',16) # 按16进制转换

def int2(x,base=2):

return int(x,base) # 定义一个int2()的函数，按2进制转换

# 上面的方式比较麻烦，我们可以用 functools.partial 来创建一个偏函数

import functools

int2=functools.partial(int,base=2) # 注意格式，前面写函数，后面改变函数的默认参数

print int2('1000000') # 按2进制转换，outout: 64

# functools.partial的作用就是，把一个函数的某些参数给固定住（也就是设置默认值），

# 返回一个新的函数，调用这个新函数会更简单。但也可以在函数调用时传入其他值

int2('1000000',base=10) # output： 1000000

# 创建偏函数时，实际上可以接收函数对象、\*args和\*\*kw这3个参数

int2 = functools.partial(int, base=2)

int2('10010')

# 上面两句与下面两句等价

kw = { base: 2 }

int('10010', \*\*kw)

#

max2 = functools.partial(max, 10)

max2(5, 6, 7)

# 实际会把10作为 \*args的一部分自动加在左边

args = (10, 5, 6, 7)

max(\*args)

# 结果都为 10

#

# 当函数的参数个数太多，需要简化时，使用functools.partial

# 可以创建一个新的函数，这个新函数可以固定住原函数的部分参数，从而在调用时更简单。

### 4.面向对象编程

#### 1.类(class)和实例

class Student(object): # 定义类是通过class关键字，class后面接类名(通常大写开头)

psss # 紧接着是object，表示该类是从哪个类继承下来的

# 如果没有合适的继承类，就使用object类，这是所有类最终都会继承的类。

bart=Student() # 通过 类名() 的方式创建实例

bart.name='bart simpson' # 给实例bart绑定一个name属性

# 由于类可以起到模板的作用，因此，可以在创建实例的时候，把一些我们认为必须绑定的属性强制填写进去

# 通过定义一个特殊的\_\_init\_\_方法，在创建实例的时候，就把name，score等属性绑上去

class Student(object):

def \_\_init\_\_(self,name,score): # \_\_init\_\_方法的第一个参数永远是self，创建的实例本身

self.name=name

self.score=score

# 有了\_\_init\_\_方法，在创建实例的时候，就不能传入空的参数了，但self不需要传

bart=Student('bart simpson',59)

#### 2.数据封装

# 面向对象编程的一个重要特点就是数据封装。

# 在上面的Student类中，每个实例就拥有各自的name和score这些数据

def print\_score(std):

print '%s: %s' % (std.name,std.score)

print\_score(bart) # 通过函数来访问实例中的数据

class Student(object):

def \_\_init\_\_(self, name, score):

self.name = name

self.score = score

def print\_score(self):

print '%s: %s' % (self.name, self.score)

bart=Student('bart simpson',59) # 数据和逻辑被“封装”起来了

bart.print\_score() # 调用很容易，但却不用知道内部实现的细节

# 封装的另一个好处是可以给Student类增加新的方法，比如get\_grade：

class Student(object):

...

def get\_grade(self):

if self.score>=90:

return 'A'

elif self.score >= 60:

return 'B'

else:

return 'C'

bart.get\_grade()

# 和静态语言不同，Python允许对实例变量绑定任何数据，也就是说

# 对于两个实例变量，虽然它们都是同一个类的不同实例，但拥有的变量名称都可能不同!

bart.score=59 # 外部代码可以自由地修改一个实例的name、score属性

# 如果要让内部属性不被外部访问，可以把属性的名称前加上两个下划线\_\_，变成一个私有变量（private）

class Student(object):

def \_\_init\_\_(self,name,score):

self.\_\_name=name

self.\_\_score=score

def print\_score(self):

print '%s: %s' % (self.\_\_name,self.\_\_score)

def get\_name(self):

return self.\_\_name # 此时只能通过这种方式获取name或score

def get\_score(self):

return self.\_\_score

def set\_score(self,score): # 修改参数的方法，严格注意缩进，tab与空格不可混用

if 0 <= score <=100:

self.\_\_score=score

else:raise ValueError('bad score')

bart=Student('bart simpson',59)

bart.\_\_name # 报错！此时无法从外部访问name或score，当然分数也就无法修改了

print bart.get\_name() # 正确的访问方式

bart.score = 85 # 输出的分数依然为59

bart.set\_score(85)

print bart.get\_score() # 这样输出的分数就变成了85

# 以一个下划线开头的实例变量名，比如\_name，这样的实例变量外部是可以访问的

# 但按照约定俗成的规定，虽然可以被访问，但请视为私有变量，不要随意访问

#### 3.继承和多态

#

# 在OPP中，当我们定义一个class时，可以从某个现有的class继承

# 新的class称为子类subclass，而被继承的class称为基类base、父类或超类super

class Animal(object):

def run(self):

print 'Animal is running'

class Dog(Animal):

pass

class Cat(Animal): # 当我们需要编写Dog和Cat类时，就可以直接从Animal类继承

pass

dog=Dog() # Dog和Cat作为Animal的子类，自动拥有了run()方法

dog.run()

class Dog(Animal): # 对子类增加一些方法，对代码做一点改进

def run(self):

print 'Dog is running...'

def eat(self):

print 'Eating meat...'

# 当子类和父类都存在相同的run()方法时，子类的run()覆盖了父类的run()

# 这就获得了继承的另一个好处：多态

# 当我们定义一个class时，我们实际上就定义了一种数据类型，和str、list等没什么不同

# 因此可以用 isinstance 来判断一个变量是否是某个类型

a=list()

b=Animal()

c=Dog()

isinstance(a,list)

isinstance(b,Animal) # 判断一个变量是否是某个类型,c既是Dog又是Animal

def run\_twice(animal):

animal.run()

animal.run()

run\_twice(Animal()) # 打印出 两个 Animal is running...

run\_twice(Dog()) # 打印出 两个 Dog is running...

# 传入的任意类型，只要是Animal类或者子类，就会自动调用实际类型的run()方法

# “开闭”原则：对扩展开放：允许新增Animal子类；

# 对修改封闭：不需要修改依赖Animal类型的run\_twice()等函数

super函数，用在派生类的方法定义里，从这个类的直接基类开始做属性检索。

#### 4.获取对象信息

#

type(123) # 利用type判断对象基本类型

type('str')

type(None)

type(abs) # 如果一个变量指向函数或者类，也可以用type()判断

type(a)

# type()函数返回的是type类型

# python 把每种type类型都定义好了常量，放在types模块里，使用之前需要先导入

import types

type('abc')==types.StringType # 以下的几条均为True

type([])=types.ListType

type(str)==types.TypeType

type(int)==types.TypeType # 所有类型本身的类型就是TypeType

# 对于class的继承关系来说，可以使用isinstance()函数。

# 如果继承关系是：object -> Animal -> Dog -> Husky

a = Animal()

d = Dog()

h = Husky()

isinstance(h,Husky) # True

isinstance(h,Dog) # True ...

isinstance('a', str) # True

isinstance('a', (str, unicode)) # 判断'a'是否是这两种类型中的一种

# 如果要获得一个对象的所有属性和方法，可以使用dir()函数，它返回一个包含字符串的list

# 类似\_\_xxx\_\_的属性和方法在Python中都是有特殊用途的，比如\_\_len\_\_方法返回长度

len('ABC')

'ABC'.\_\_len\_\_() # 3 这两条语句实现的功能相同

'ABC'.lower() # 返回小写的字符串

# 通过内置的一系列函数，我们可以对任意一个Python对象进行剖析，拿到其内部的数据

def readImage(fp):

if hasattr(fp,'read'):

return readData(fp)

return None

# 我们希望从文件流fp中读取图像，我们首先要判断该fp对象是否存在read方法

# 如果存在，则该对象是一个流，如果不存在，则无法读取。

# 注意，在Python这类动态语言中，有read()方法，不代表该fp对象就是一个文件流，

# 它也可能是网络流，也可能是内存中的一个字节流，

# 但只要read()方法返回的是有效的图像数据，就不影响读取图像的功能。

#### 5.面向对象编程的高级特性

# 数据封装、继承和多态只是面向对象程序设计中最基础的3个概念。

# 在Python中，面向对象还有很多高级特性，允许我们写出非常强大的功能。

# https://www.liaoxuefeng.com/wiki/001374738125095c955c1e6d8bb493182103fac9270762a000/001386820058291028118ddeefc4de7860a8e48b9942e9b000

# 在面向对象高级编程中，会提供以下内容的学习

# 使用\_\_slots\_\_

# 使用@property

# 多重继承、定制类、元类

### 5. 模块 module

# 随着程序代码越写越多，维护也成为了一个大问题

# 为了编写可维护的代码，我们把很多函数分组，分别放到不同的文件里，

# 这样每个文件包含的代码就相对较少，在python中，一个.py文件就称为一个模块module

# 使用模块最大的好处是大大提高了代码的可维护性

# 其次，当一个模块填写完毕，就能被其他地方引用，包括python内置的模块和来自第三方的模块

# 使用模块还可以避免函数名和变量名冲突。相同名字的函数和变量完全可以分别存在不同的模块中

# 注意，尽量不要与内置函数名字冲突，python所有内置函数见此：http://docs.python.org/2/library/functions.html

# 如果不同的人编写的模块名相同怎么办？为避免模块名冲突，Python引入了按目录来组织模块的方法，称为包（Package）。

# abc.py就是一个名字叫abc的模块，假设abc这个模块名字与其他模块冲突了，于是可以通过包来组织模块

# 方法是选择一个顶层包名，比如mycompany,abc.py存放在mycompany下面

# 引入了包以后，只要顶层的包名不与别人冲突，那所有模块都不会与别人冲突

# 此时，abc.py 模块的名字就变为了mycompany.abc

# 注意：每一个包目录下都有 \_\_init\_\_.py 的文件，否则，python就把这个目录当成普通目录，而不是一个包

# 类似的，可以有多级目录，文件 www.py 的模块名就是 mycompany.web.www

#### 1.使用模块

#

# 以内建的sys模块为例，编写一个hello的模块

#!/usr/bin/env python # 标准注释：可以让hello.py文件直接在unix/linux/mac上运行

# -\*- coding: utf-8 -\*- # .py文件使用标准UTF-8编码,-\*-可以省去

' a test module'

\_\_author\_\_='patrik yu' # 写上作者大名

import sys # 导入sys模块

# sys模块有一个argv变量，用list存储了命令行的所有参数

# argv至少有一个元素，因为第一个参数永远是该.py文件的名称

# 例如：运行python hello.py获得的sys.argv就是['hello.py']

# 运行python hello.py Michael获得的sys.argv就是['hello.py', 'Michael']

def test():

args=sys.argv

if len(args)=1:

print 'hello,world'

elif len(args)=2:

print 'hello,%s' % args[]

else:

print 'too many arguments'

if \_\_name\_\_=='\_\_main\_\_': # 常用于运行测试

test()

# 当我们在命令行运行hello模块文件时，Python解释器把一个特殊变量\_\_name\_\_置为\_\_main\_\_

# 如果在其他地方导入该hello模块时，if判断为False，不执行test()

# 用命令行运行hello.py,会执行test()函数；

# 如果启动Python交互环境，再导入hello模块，import hello，不会执行test()函数

# 导入模块时，还可以使用别名，这样，可以在运行时根据当前环境选择最合适的模块。

# Python标准库一般会提供StringIO和cStringIO两个库，它们的接口和功能是一样的，但是cStringIO是C写的，速度更快

try:

import cStringIO as StringIO

except ImportError: # 导入失败会捕获到 ImportError

import StringIO

# 这样就可以优先导入cStringIO。如果有些平台不提供cStringIO，还可以降级使用StringIO

# 导入cStringIO时，用import ... as ...指定了别名StringIO，因此，后续代码引用StringIO即可正常工作。

# 还有类似simplejson这样的库，在Python 2.6之前是独立的第三方库，从2.6开始内置

try:

import json # python >= 2.6

except ImportError:

import simplejson as josn # python <= 2.5

# 作用域：有的函数和变量我们希望给别人使用，有的函数和变量希望仅仅在模块内部使用。在Python中，通过\_前缀来实现的。

# 正常的函数和变量名是公开的（public），可以被直接引用，比如：abc，x123，PI等

# 类似\_\_xxx\_\_这样的变量是特殊变量，可以被直接引用，但是有特殊用途，比如上面的\_\_author\_\_

# hello模块定义的文档注释也可以用特殊变量\_\_doc\_\_访问，我们自己的变量一般不要用这种变量名

# 类似\_xxx和\_\_xxx这样的函数或变量就是非公开的（private），不应该被直接引用，比如\_abc，\_\_abc等

# private函数和变量“不应该”被直接引用，而不是“不能”被直接引用，

# 因为Python并没有一种方法可以完全限制访问private函数或变量，囧

#

def \_private\_1(name):

return 'hello,%s' % name

def \_private\_2(name):

return 'hi,%s' % name

def greeting(name): # 在模块里公开greeting()函数

if len(name)>3:

return \_private\_1(name) # 而把内部逻辑用private函数隐藏

else:

return \_private\_2(name) # 这是一种非常有用的代码封装和抽象的方法

#### 2.安装第三方模块

#

# 在Python中，安装第三方模块，是通过setuptools这个工具完成的

# Python有两个封装了setuptools的包管理工具：easy\_install和pip。目前官方推荐使用pip

# 让我们来安装一个第三方库——Python Imaging Library，这是Python下非常强大的处理图像的工具库。

# 要安装一个第三方库，必须先知道该库的名称，可以在官网或者pypi上搜索，比如Python Imaging Library的名称叫PIL

pip install PIL # 耐心等待下载并安装后，就可以使用PIL了

# 有了PIL，处理图片易如反掌。随便找个图片生成缩略图

import Image

im=Image.open('test.png')

print im.format,im.siza,im.module # output: PNG (400, 300) RGB

im.thumbnail((200,100))

im.save('thumb.jpg','JPEG')

# MySQL的驱动：MySQL-python，用于科学计算的NumPy库：numpy，用于生成文本的模板工具Jinja2

import sys

sys.path # 默认情况下，Python解释器会搜索当前目录所有模块，搜索路径存放在sys模块的path变量中

# 如果我们要添加自己的搜索目录，有两种方法：

import sys

sys.path.append('/Users/patrik/mypy') #方法一：直接修改sys.path，添加要搜索的目录

# 此方法在运行时修改，运行结束后失效

# 方法二：是设置环境变量PYTHONPATH，该环境变量的内容会被自动添加到模块搜索路径中。

# 设置方式与设置Path环境变量类似

#### 3.使用\_\_future\_\_

#

# Python的每个新版本都会增加一些新的功能，或者对原来的功能作一些改动。有些改动是不兼容旧版本的

# 比如2.x里的字符串用'xxx'表示str，Unicode字符串用u'xxx'表示unicode，而在3.x中，所有字符串都被视为unicode

# Python提供了\_\_future\_\_模块，把下一个新版本的特性导入到当前版本，于是我们就能在当前版本中测试一些新版本的特性

# 为了适应Python 3.x的新的字符串的表示方法，在2.7版本的代码中，

# 可以通过unicode\_literals来使用Python 3.x的新的语法

# still running on python 2.7

from \_\_future\_\_ import unicode\_literals

print '\'xxx\'is unicode?',isinstance('xxx',unicode)

# Python 2.x中，对于除法有两种情况，如果是整数相除，结果仍是整数，余数会被扔掉，这种除法叫“地板除”

# 要做精确除法，必须把其中一个数变成浮点数

# 而在Python 3.x中，所有的除法都是精确除法，地板除用//表示

# 如果你想在Python 2.7的代码中直接使用Python 3.x的除法，可以通过\_\_future\_\_模块的division实现

from \_\_future\_\_ import division

print '10/3=',10/3

print '10.0/3=',10.0/3

print '10//3=',10//3

### 6.错误、调试和测试

# Python内置了一套异常处理机制，来帮助我们进行错误处理。pdb可以让我们以单步方式执行代码。

#

#### 1.错误处理

# 在程序运行的过程中，如果发生了错误，可以事先约定返回一个错误代码，但是这样很不方便

# 所以高级语言通常都内置了一套try...except...finally...的错误处理机制，Python也不例外

try:

print 'try...'

r=10/0

print 'result:',r

except ZeroDivisionError,e:

print 'except:',e

finally:

print 'finally...'

print 'END'

# 当我们认为某些代码可能会出错时，就可以用try来运行这段代码

# 如果执行出错，则后续代码不会继续执行，而是直接跳转至错误处理代码，即except语句块

# 执行完except后，如果有finally语句块，则执行finally语句块，至此，执行完毕。

# 如果没有错误发生，except语句块不会被执行，但是finally如果有，则一定会被执行

# 错误有很多种类，如果发生了不同类型的错误，应该由不同的except语句块处理

try:

print 'try...'

r = 10 / int('a')

print 'result:', r

except ValueError, e:

print 'ValueError:', e

except ZeroDivisionError, e:

print 'ZeroDivisionError:', e

else:

print 'no error!'

finally:

print 'finally...'

print 'END'

# int()函数可能会抛出ValueError，所以我们用一个except捕获ValueError，用另一个except捕获ZeroDivisionError

# Python的错误其实也是class，所有的错误类型都继承自BaseException

# 所以在使用except时需要注意的是，它不但捕获该类型的错误，还把其子类也“一网打尽”

# 常见的错误类型和继承关系见此：https://docs.python.org/2/library/exceptions.html#exception-hierarchy

# 使用try...except捕获错误还有一个巨大的好处，就是可以跨越多层调用

def foo(s):

return 10 / int(s)

def bar(s):

return foo(s) \* 2

def main():

try:

bar('0')

except StandardError, e:

print 'Error!'

finally:

print 'finally...'

# 函数main()调用foo()，foo()调用bar()，结果bar()出错了，这时，只要main()捕获到了，就可以处理

#

# 调用堆栈：如果错误没有被捕获，它就会一直往上抛，最后被Python解释器捕获，打印一个错误信息，然后程序退出。

#

# 记录错误：Python内置的logging模块可以非常容易地记录错误信息

# 同样是出错，但程序打印完错误信息后会继续执行，并正常退出

# 通过配置，logging还可以把错误记录到日志文件里，方便事后排查。

#

# 抛出错误：错误是class，捕获一个错误就是捕获到该class的一个实例。错误不是凭空产生的，而是有意创建并抛出的

# 如果要抛出错误，首先根据需要，可以定义一个错误的class，选择好继承关系

# 然后，用raise语句抛出一个错误的实例，执行，可以最后跟踪到我们自己定义的错误

# 如果可以选择Python已有的内置的错误类型（比如ValueError，TypeError），尽量使用Python内置的错误类型

#### 2.调试

# 第一种方法简单直接粗暴有效，就是用print把可能有问题的变量打印出来看看

# 凡是用print来辅助查看的地方，都可以用断言（assert）来替代

# 终极武器：把print替换为logging，和assert比，logging不会抛出错误，而且可以输出到文件

# 第4种方式是启动Python的调试器pdb，让程序以单步方式运行，可以随时查看运行状态

# 第5个方法也是用pdb，但是不需要单步执行，我们只需要import pdb，然后在可能出错的地方放一个pdb.set\_trace()，就可以设置一个断点

# 最方便的方法：使用IDE，推荐用pycharm

#### 3.单元测试（未学习）

#### 4.文档测试（未学习）

### 7.正则表达式

# 正则表达式是一种用来匹配字符串的强有力的武器。它的设计思想是用一种描述性的语言来给字符串定义一个规则，

# 凡是符合规则的字符串，我们就认为它“匹配”了，否则，该字符串就是不合法的。

# 所以我们判断一个字符串是否是合法的Email的方法是：

# 1.创建一个匹配Email的正则表达式；

# 2.用正则表达式去匹配用户的输入来判断是否合法。

# 在正则表达式中，如果直接给出字符，就是精确匹配。用\d可以匹配一个数字，\w可以匹配一个字母或数字

# '00\d'可以匹配'007'，但无法匹配'00A'。'\w\w\d'可以匹配'py3'

# 'py.'可以匹配'pyc'、'pyo'、'py!'等等。

# 匹配变长的字符，在正则表达式中，用\*表示任意个字符（包括0个），用+表示至少一个字符，用?表示0个或1个字符，

# 用{n}表示n个字符，用{n,m}表示n--m个字符：

# \d{3}\s+\d{3,8}

# 1. \d{3}表示匹配3个数字，例如'010'

# 2. \s可以匹配一个空格（也包括Tab等空白符），所以\s+表示至少有一个空格，例如匹配' '，' '等

# 3. \d{3,8}表示3-8个数字，例如'1234567'

# 综合起来，上面的正则表达式可以匹配以任意个空格隔开的带区号的电话号码。

# 如果要匹配'010-12345'这样的号码呢？由于'-'是特殊字符，在正则表达式中，要用'\'转义

# 所以，上面的正则是\d{3}\-\d{3,8}

# 要做更精确地匹配，可以用[]表示范围：

# [0-9a-zA-Z\\_]可以匹配**一个**数字、字母或者下划线

# [a-zA-Z\\_][0-9a-zA-Z\\_]\*可以匹配由字母或下划线开头，后接任意个由一个数字、字母或者下划线组成的字符串，

# 也就是Python合法的变量

# [a-zA-Z\\_][0-9a-zA-Z\\_]{0, 19}更精确地限制了变量的长度是1-20个字符（前面1个字符+后面最多19个字符）

# A|B可以匹配A或B，所以(P|p)ython可以匹配'Python'或者'python'

# ^表示行的开头，^\d表示必须以数字开头。

# $表示行的结束，\d$表示必须以数字结束。

# Python提供re模块，包含所有正则表达式的功能

# 由于Python的字符串本身也用\转义，所以要特别注意：

s = 'ABC\\-001' # Python的字符串

# 对应的正则表达式字符串变成：

# 'ABC\-001'

# 强烈建议使用Python的r前缀，就不用考虑转义的问题了：

s = r'ABC\-001' # Python的字符串

# 对应的正则表达式字符串不变：

# 'ABC\-001'

# 1.match()方法判断是否匹配，如果匹配成功，返回一个Match对象，否则返回None

# 2.切分字符串：

re.split(r'[\s\,\;]+', 'a,b;; c d')

# 3.分组：除了简单地判断是否匹配之外，正则表达式还有提取子串的强大功能。用()表示的就是要提取的分组（Group）。

# 4.贪婪匹配：正则匹配默认是贪婪匹配，也就是匹配尽可能多的字符。

# 5.编译

<https://www.liaoxuefeng.com/wiki/001374738125095c955c1e6d8bb493182103fac9270762a000/001386832260566c26442c671fa489ebc6fe85badda25cd000>

### 8.NumPy

#### 1.array与matrix

mm=array([1,1,1]) pp=array([[1,2,3],[1,1,1]])

mm\*2,mm\*\*2(平方操作) 均对元素进行操作

采用列表的访问方式 mm[1] mm[0][1] 也可采用矩阵访问方式 mm[0,1]

当把两个数组乘起来的时候，两个数组的元素将对应相乘

ss=mat([1,2,3]) rr=mat([[1,2,3],[1,1,1]])

将列表转换为Numpy矩阵： pyList=[5,11,1605];mat(pyList)

矩阵的乘法要满足相乘条件，若要把每个元素对应相乘，采用 multiply(m,s)的方法

.T 实现对矩阵的转置，.I 求逆

矩阵和数组还有很多有用的地方：

比如排序：mm.sort() 原地排序，默认升序

dd.argsort() 得到矩阵中每个元素的排序序号

计算矩阵的均值 dd.mean()

矩阵的快速索引方法 jj[1,:] jj[1,0:2]

### 9. Matplotlib

#### 1.plot操作

#创建数据

x = linspace(-5, 5, 100)

y1 = sin(x)

y2 = cos(x)

#创建figure窗口

plt.figure(num=3, figsize=(8, 5))

#画曲线1

plt.plot(x, y1)

#画曲线2

plt.plot(x, y2, color='blue', linewidth=5.0, linestyle='--')

#设置坐标轴范围

plt.xlim((-5, 5))

plt.ylim((-2, 2))

#设置坐标轴名称

plt.xlabel('xxxxxxxxxxx')

plt.ylabel('yyyyyyyyyyy')

#设置坐标轴刻度

my\_x\_ticks = arange(-5, 5, 0.5)

my\_y\_ticks = arange(-2, 2, 0.3)

plt.xticks(my\_x\_ticks)

plt.yticks(my\_y\_ticks)

#显示出所有设置

plt.show()

# 另外一个例子：

x=linspace(-pi,pi,256,endpoint=True)

# 起点 终点 样本点数

C,S= cos(x), sin(x)

plt.xlim(x.min()\*1.1, x.max()\*1.1)

# 设置横轴和纵轴的界面

plt.ylim(C.min()\*1.1, C.max()\*1.1)

plt.plot(x,C,color='red',linewidth=2.5,linestyle='-',label=r'$cos(t)$') # 红线标记为cos(t)

plt.plot(x,S,color='blue',linewidth=2.5,linestyle='-',label=r'$sin(t)$')

plt.legend(loc='upper left',frameon=False) # 标注在左上角

plt.show()

#### 2.用scatter绘制散点图

import matplotlib.pyplot as plt # pyplot是matplotlib里最常用的作图模块,将matplotlib缩写为plt

fig=plt.figure() # 创建figure窗口

ax1=fig.add\_subplot(221)

# 创建 x 为 0 和 之间的 200 个等间距值。创建 y 为带随机干扰的余弦值。然后，创建一个散点图

x = linspace(0,3\*pi,200);

y = cos(x) + random.rand(1,200);

ax1.scatter(x,y)

# 使用大小不同的圆圈创建一个散点图。以平方磅为单位指定大小

# x、y 和 sz 中的相应元素确定每个圆圈的位置和大小。要按照相同的面积绘制所有圆圈，请将 sz 指定为数值标量

ax2=fig.add\_subplot(222)

x = linspace(0,3\*pi,200)

y = cos(x) + random.rand(1,200)

#设置标题

ax2.set\_title('Scatter Plot')

#设置X轴标签

plt.xlabel('X')

#设置Y轴标签

plt.ylabel('Y')

#设置图标

plt.legend('x2')

#画散点图

ax2.scatter(x,y,c = 'r',marker = 'o')

plt.show()

另一个例子;

fig=plt.figure() # 创建figure窗口

ax=fig.add\_subplot(111) # 不能通过空figure绘图。必须使用add\_subplot()创建一个或多个subplot，

# 111创建了1\*1个图，ax在1中绘制

# ax.scatter(x,y,s=20,color='b',marker='o',cmap=None,norm=None,vmin、vmax、alpha、linewidths、verts)

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

# 横坐标 纵坐标 点的大小，\*15得到的点大一点 颜色，同一类的颜色相同

# 这样可以同时画出所有的点，但是没有办法写标签注释，要写标签注释的话可以用下面的方法

idx\_1 = nonzero(array(datingLabels)==1) # nonzero就和matlab里的find函数差不多，注意datingLabels必须先转为array！这样才能使用nonzero

p1 = plt.scatter(datingDataMat[idx\_1,2], datingDataMat[idx\_1,1], marker = 'x', color = 'm', label='dislike', s = 15)

idx\_2 = nonzero(array(datingLabels)==2)

p2 = plt.scatter(datingDataMat[idx\_2,2], datingDataMat[idx\_2,1], marker = '+', color = 'c', label='justsoso', s = 30)

idx\_3 = nonzero(array(datingLabels)==3)

p3 = plt.scatter(datingDataMat[idx\_3,2], datingDataMat[idx\_3,1], marker = 'o', color = 'r', label='charming', s = 45)

plt.legend(loc = 'upper left')

ax.set\_title('class')

plt.xlabel('fly')

plt.ylabel('time on games')

plt.legend()

plt.show()

### 10.Tkinter库创建GUI

机器学习实战P176 

## 常用机器学习算法及代码

### 算法介绍

#### 1.k-近邻算法

监督算法，需要保存所有训练样本集，输入没有标签的新数据后，将新数据的每个特征与样本集中的数据对应的特征进行比较，计算他们之间的距离，按照距离递增排序，取最接近的k个数据并统计这些数据中出现次数最多的标签，认为此新数据应定义为此标签。

**优点**：精度高，对异常值不敏感，无数据输入假定

**缺点**：计算复杂度高，空间复杂度高，无法给出数据的内在含义

伪代码:

1.计算已知类别数据集中的点与当前点之间的距离

2.按照距离递增次序排序

3.选取与当前点距离最小的k个点

4.确定前k个点所在类别的出现频率

5.返回前k个点出现频率最高的类别作为当前点的预测分类

总结：K-近邻是监督学习中分类数据最简单有效的算法。K-近邻算法是基于实例的学习，使用算法时我们必须有接近实际数据的训练样本数据。它必须保存全部数据集，若训练数据集很大，必须使用大量的存储空间。另一个缺陷在于，它无法给出任何数据的基础结构信息，无法给出数据的内在含义，因此我们无法知晓平均实例样本和典型实例样本具有什么特征。下一章我们将使用概率测量方法处理分类问题，该算法可以解决这个问题。

实际使用这个算法时，算法的执行效率并不高，因为算法需要为每个测试向量做2000次距离计算，每个距离计算包括了1024个维度浮点计算，总计要执行900次。此外，我们还需要为测试向量准备2MB的存储空间。

是否存在一种算法减少存储空间和计算时间的开销呢？使用k近邻算法的优化版——k决策树吧！

#### 2.决策树

监督学习算法。决策树的主要优势在于数据形式非常容易理解。决策树的一个重要任务是为了理解数据中所蕴含的知识信息，因此决策树可以使用不熟悉的数据集合，并从中提取出一系列规则，这些机器根据数据集创建规则的过程，就是机器学习的过程。

**优点**：计算复杂度不高，输出结果易于理解，对中间值的缺失不敏感，可以处理不相关特征数据

**缺点**：可能会产生过度匹配问题

我们需要解决的第一个问题就是，当前数据集上哪个特征在划分数据分类时起决定性作用。为了找到决定性的特征，划分出最好的结果，首先我们必须评估每个特征。划分数据集的大原则是：将无序的数据变得更加有序。在划分数据集之前之后信息发生的变化称为信息增益（信息增益=熵的减少量），计算每个特征值划分数据集获得的信息增益，获得信息增益最高的特征就是最好的选择。不断重复此过程，直到同一个分支下的所有数据子集的数据属于同一个类别（即标签值相同）。  
 注意：树构造算法只适用于标称型数据，因此数值型数据必须离散化

如今决策树是一种简单但是广泛使用的分类器。常见的算法包括 CART (Classification And Regression Tree)、ID3、C4.5、随机森林 (Random Forest) 等。这里采用ID3算法划分数据集，该算法处理如何划分数据集，何时停止划分数据集。如果依据某个属性划分数据将会产生4个可能的值，我们就把数据划分为四块，并创建四个不同的分支。

度量信息的方法（度量数据集的无序程度）：信息熵，公式见p35，详细介绍见数学之美p86，第六章。熵越高，则混合的数据也越多，如果在数据集中添加更多的分类，可以看到熵的值也增大了。另一个度量集合无序程度的方法是基尼不纯度（Gini inpurity），简单地说就是从一个数据集中随机选取子项，度量其被错误分类到其他分组里的概率。

递归构建决策树：第一次划分之后，数据将被向下传递到树分支的下一个节点，在这个节点上，我们可以再次划分数据。细节见p41创建树的函数代码，这是一个经典的迭代。递归结束的条件是：程序遍历完所有划分数据集的属性，或者每个分支下的所有实例都具有相同的分类（即标签相同）。其他的决策树算法如C4.5和CART，这些算法在运行时并不总是在每次划分分组时都会消耗特征。由于特征数目并不是在每次划分数据分组时都减少，因此这些算法在实际使用过程中可能会引起一定的问题（这里没考虑这个问题）。如果数据集已经处理了所有属性，但是类标签依然不是唯一的，此时我们需要决定如何定义该叶子节点，一般采用多数表决的方法来决定该叶子节点的分类。

依靠训练数据构造了决策树之后，我们可以把它应用于实际数据的分类。在执行数据分类时，需要使用决策树以及用于构造决策树的标签向量。然后，程序比较测试数据与决策树上的数值，递归执行该过程直到进入叶子节点，最后将测试数据定义为叶子节点所属的类型。

{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}

上面为构造出的决策树。

使用算法：决策树的存储

构造决策树是很耗时的任务，然而用创建好的决策树解决分类问题，则可以很快完成

这就要求使用pickle序列化对象，序列化对象可以在磁盘上保存对象，并在需要的时候读取出来，任何对象都可以序列化操作

def storeTree(inputTree,filename):  
 import pickle  
 fw=open(filename,'w')  
 pickle.dump(inputTree,fw)  
 fw.close()  
  
def grabTree(filename):  
 import pickle  
 fr=open(filename)  
 return pickle.load(fr)  
# 可以把上面两个函数当做 模块 来使用

storeTree(myTree,'classifierStorage.txt')

# 将myTree 存储为 'classifierStorage.txt'文件

print grabTree('classifierStorage.txt')

# 打开 'classifierStorage.txt'文件

示例：使用决策树预测隐形眼镜类型

隐形眼镜数据集是非常著名的数据集，它包含很多患者眼部状况的观察条件以及医生推荐的隐形眼镜类型（包含硬材质，软材质，不适合戴隐形眼镜）

将隐形眼镜数据集作为训练样本，解析tab键分隔的数据，需要提供训练样本以及特征名称，训练样本要符合以下要求：

1.数据必须是一种由列表元素组成的列表，而且所有的列表元素都要具有相同的数据长度

2.数据的最后一列或每个实例的最后一个元素是当前实例的类别标签

fr=open('lenses.txt')  
lenses=[inst.strip().split('\t') for inst in fr.readlines()]   
# 每一行数据的各个特征之间用制表符来分隔  
# 得到的是以 ，连接的各个特征，各个样本均以list形式表示  
lensesLabels=['age','prescript','astigmatic','tearRate']  
lensesTree=createTree(lenses,lensesLabels) # 训练算法  
treePlotter.createPlot(lensesTree) # 画出树图

在python中使用Matplotlib注释绘制树形图

matplotlib提供了一个非常有用的注释工具annotations,它可以在数据图形上添加文本注解。

具体代码见

总结：以上构造的决策树非常好地匹配了实验数据，然而这些匹配选项可能太多了，我们将这种问题称之为过度匹配（overfitting）。为了减少过度匹配的问题，我们可以裁剪决策树，去掉一些不必要的叶子节点。如果叶子节点只能增加少许信息，则可以删除该节点，把它并入其他叶子节点中。第九章将会学习相关的决策树构造算法CART（C4.5和CART是最流行的两种决策树算法）。本章使用的算法称之为ID3，它无法直接处理数值型数据，尽管我们可以通过量化的方法将数值型数据转换为标称型数据，但是如果存在太多的特征划分，ID3算法仍然会面临其他问题。

K-近邻以及决策树都是结果确定的分类算法，数据实例最终会被明确划分到某个分类中。

拓展：简单介绍一下C4.5、随机森林 (Random Forest)

C4.5：C4.5其实就是ID3的扩展，最简单的决策树形式ID3用来处理离散值输入和输出，于是人们扩展了决策树的功能，并加入了其他的扩展，衍生出了C4.5。C 4.5比ID3多了一些特性：

1、可用于对连续型输入或输出进行拟合。虽说如此，其实也只是把连续的值划分为几个区间来输入而已。

2、可处理缺少某些属性的数据。收集的数据常常会有某些属性值是缺失的，就像医院里统计的病人的各种检测数据，总会有A做了抽血化验没有尿检，B做了尿检却没有抽血化验。这个时候信息增益就很难计算，所以我们会主动给这些数据“补上”缺失的属性。一般有三种方法：①赋给他结点n下的训练样例中该属性的最常见值②赋给他结点n下被分类为他所属的类里该属性的最常见值。③按照概率划分。按照结点n下该属性在已知数据中出现的概率划分给其他未知的实例。

3、增加了对过度拟合的处理：错误率后修剪法和规则后修剪法。矫正过度拟合最常采用验证集合来调整。①错误率后修剪法：对生成的决策树自下而上，每次删除一个节点，然后在验证集合上进行验证，如果修剪后的决策树在验证集合上得到了更小的错误率，那就说明这次修剪是有效的，剔除那个节点，否则决策树不变，知道判断完所有节点。②规则后修剪：将各个节点表示成if-then的规则化表示再进行修剪。如上面提到的那棵决策树，最左一条支路用规则来表示可以写成：iIF (Outlook=sunny) and (Humidity=high) ,THEN PlayTennis=No，像这样将所有的支路表示成规则后，每次剔除一个规则就统计在验证集合上的错误率情况，直到得到最优的决策树。

一般来说第二种方法较常采用，因为规则化后的形式比较有利于人的理解，而且每次剔除的规则只是某个属性的一个值，但是第一种方法却剔除了一个属性下的所有属性值，这是不太好的。

4、处理代价不同的属性：某些属性不好采集，或者属性的采集需要付出很高的成本代价(人力或者金钱)时，我们通常只在需要可靠分类时才统计他们，当不需要时会尽量选用代价低的属性来进行决策树的生成。

随机森林 (Random Forest): 随机森林并不是独立于ID3，C4.5，cart之外的算法，他结合了数学上的随机抽样，运用自助法(boot-strap)重采样，来达到一种更好的拟合。下面介绍自助法中的bagging模式，步骤如下：

①假设目前训练数据D有N个实例，M个特征，我们采用放回抽样抽出n个数据，m个特征组成一个训练数据D1，再用同样的方法抽出n、m组成D2，然后D3......Dn

②分别用这n组训练数据，如采用C4.5分别生成一个决策树。

③将每个决策树的决策汇总，选择当中占比最高的决策作为整个森林的决策结果。（这种方法使用最多）

或

③用验证集合对n个决策树进行投票，选中表现最好的决策树。(投票的意思应该是看谁在验证集合上分类出错率最小)

另一种自助法采用模式是Adaboosting，有兴趣可以百度下，他主要对每个实例分配一个概率来抽取。

由整个森林的投票来决定结果可以避免单棵树过拟合的情况，使得决策更加合理，随机森林近年来在各种数据挖掘大赛前几名中出现的频率极高，是一个很有潜力的算法。

对于第二个③，因为采用了随机抽样来生成训练数据，所以n个数据集里有很大可能出现一个不包含噪声或噪声很少的数据集Di，由他生成的决策树将在后来的投票中具有很高的票数。因此随机森林的方法对噪声有很好的鲁棒性，而且能够很大程度上避免过度拟合，因为我们总是选取在验证集合上表现最好的一棵决策树。

python的scikit-learn包里封装了随机森林，感兴趣的同学可以了解下。scikit-learn里封装了很多分类器，帮使用者节省了很多时间，scikit-learn简单到了甚至被称作了toy play工具包了。

#### 3.基于概率论的分类方法：朴素贝叶斯

分别计算数据点(x,y)属于类别1的概率以及属于类别2的概率，选择具有更高概率的决策作为该数据点的类别

优点：在数据较少的情况下仍然有效，可以处理多类别问题

缺点：对于输入数据的准备方式较为敏感，仅适用于标称型数据。由于使用了样本属性独立性的假设，所以如果样本属性有关联时其效果不好

由统计学可知，如果每个特征需要N个样本，那么对于10个特征就需要N的10次方个样本。（假设1）特征之间相互独立，那么样本数就可以减少为1000\*N。独立，即一个特征或者单词出现的可能性与它和其他单词相邻没有关系。（假设2）每个特征同等重要。尽管上述假设存在瑕疵，但朴素贝叶斯的实际效果却很好。

拓展：看看数学之美里的基于统计方面的知识，马尔科夫假设

接下来我们应用朴素贝叶斯进行文档分类，从文本中构造词向量有两种方法：

1. 将每个词的出现与否作为一个特征，这是词集模型(set-of-words model)
2. 如果记录下每个词出现的次数，这种方法被称为词袋模型(bag-of-words model),词袋模型可以收集到更多有用的信息

见P60，利用条件独立性假设来简化概率的计算

计算每个词在侮辱性文档中出现的次数/它在文档中出现的总次数，即为这个词为侮辱性词汇的概率。这里采用的是向量的方法来操作。p1Vect为侮辱性词汇的概率向量，即每个词属于侮辱性词汇的概率都放在这个向量中。

待分类特征向量\*p0Vect, 待分类特征向量\*p1Vect, pAbusive 分别对应着课本中的 p(w|c0), p(w|c1),p(c0) p(c1)=1-p(c0)

# 注意两点:1.利用贝叶斯分类器对文档进行分类时，要计算多个概率的乘积以获得文档（样本）属于某个类别的概率，如果其中一个概率值为0，那么最后的乘积也为0，因此所有词的出现数初始化为1，并将分母初始化为2

# 2.计算多个概率的乘积时，由于大部分因子都非常小，所以程序会下溢出。解决方法是对乘积取自然对数。

p1Vect = log(p1Num / p1Denom)

p1 = sum(vec2Classify \* p1Vec) + log(pClass1)   
# log(a)+log(b)=log(a\*b),p1就是P(W0,W1...Wn|Ci)\*P(Ci)

# \*相乘指的是元素之间的相乘

vec2Classify 是待分类的词汇特征向量，p1即该样本属于侮辱性样本的概率（除以p（w）），直接比较p(1)和p(0)的大小即可知道分类。

在切分词汇的过程中，可能需要移除停用词，或者花大量时间对切分器进行优化。

for pairW in top30Words:  
 if pairW[0] in vocabList: vocabList.remove(pairW[0])

# 去掉vocabList中的top 30 words

# 语言中大部分都是冗余和结构上的辅助词，如的地得。这些高频词不能作为特征词，移除top30或其他可以通过比较准确率来适当选择

# 另一个常用的方法是不仅移除高频词，而且还同时从某个预定的词表（stop word list）中移除结构上的辅助词，的地得之类的

小结：对于分类来说，使用概率有时要比使用硬规则更为有效。可以通过特征之间的条件独立性假设，降低对数据量的需求。尽管条件独立性假设并不正确，但是朴素贝叶斯仍然是一种有效的分类器。下溢出问题可以通过对概率取对数来解决。词袋模型在解决文档分类问题上比词集模型有所提高。

#### 4.Logistic回归（分类算法）

这是首次接触到最优化算法。利用Logistic回归进行分类的主要思想是：根据现有数据对分类边界线建立回归公式，以此进行分类。这里的“回归”一词来源于最佳拟合，表示要找到最佳拟合参数集。由于需要进行距离计算，因此**要求数据类型为数值型**。

优点：计算代价不高，易于理解与实现

缺点：容易欠拟合，分类精度可能不高

将z=x代入Sigmoid函数中，进而得到一个范围在0~1之间的数值。任何大于0.5的数据被分入1类，小于0.5即被分入0类。所以逻辑回归也可以被看成是一种概率估计。

这里介绍的第一个优化算法叫做梯度上升法：要找到某函数的最大值，最好的方法是沿着该函数的梯度方向探寻。梯度下降算法和这里的梯度上升算法是一样的，只是公式中的加法需要变成减法。梯度上升算法用来求函数的最大值，而梯度下降算法用来求函数的最小值。

W初始权重矩阵设为1。对于逻辑回归，x\*error得到的就是梯度https://blog.csdn.net/kaka19880812/article/details/46993917 有推导过程

画出决策边界，决策边界即z=x=0的地方，此时Sigmoid函数=0.5

拟合曲线为0 = w0\*x0+w1\*x1+w2\*x2, 故x2(y) = (-w0\*x0-w1\*x1)/w2

一次处理所有数据被称为“批处理”，，随机梯度上升算法：一次仅用一个样本点来更新回归系数。这样就大大减小了计算复杂度。由于可以在新样本到来时对分类器进行增量式更新，因此随机梯度上升算法是一个在线学习算法。迭代过程显示，回归系数在大的波动停止以后，还有一些小的周期性波动，这是因为存在一些不能正确分类的样本点（数据集并非线性可分），在每次迭代时会引发系数的剧烈改变。我们期望算法能避免来回波动，更快地收敛到某个值。可以通过增加两处代码来改进：1.alpha（学习率）每次迭代时都会进行调整，这会缓解数据波动或者高频波动，并且加快了收敛的速度。虽然alpha会随着迭代次数不断减小，但永远不会为0，而且还要保证alpha不是严格下降的， 避免参数的严格下降也常见于模拟退火算法等其他优化算法中。2.通过随机选取样本来更新回归系数，这种方法将减少周期性的波动。

小结：Logistic回归的目的是寻找一个非线性函数Sigmoid的最佳拟合参数，求解过程可以由最优化算法来完成。最优化算法中最常用的就是梯度上升算法。

#### 5.支持向量机SVM

SVM可能是现成的（不加修改直接使用）的最好的分类器。SVM有很多实现，但本章只关注其中最流行的一种实现，即序列最小优化（SMO）算法。

优点：泛化错误率低（泛化误差常常用测试误差来表示，推广到新数据的优劣程度），计算开销不大，结果易解释

缺点：对参数调节和核函数的选择敏感，原始分类器不加修改仅适用于二类问题

将数据集分隔开来的直线（数据的N-1维）称为分隔超平面，也就是分类的决策边界。是否应该最小化数据点到分隔超平面的距离来求最佳直线？不，这并非最佳方案。我们希望找到离分隔超平面最近的点，确保它们离分隔面的距离（间隔margin）尽可能远。支持向量（support vector）就是离分隔超平面最近的那些点。分隔超平面的形式可以写成 x+b ，也可以计算出点A到分隔超平面的距离。

用一个类似单位阶跃的函数对x+b作用，得到f(x+b)，当u<0时f(u)输出-1，反之则输出+1。+-1仅仅相差一个符号，可以通过一个统一的公式表示点到分隔超平面的距离。优化目标函数的推导及KKT条件，参考李航《统计学习方法》。补充：吴恩达所说的是另一种优化目标函数，即铰链损失函数（Hinge Loss），这是SVM优化目标函数的另一种推导方式，从逻辑回归推导过来的。

接下来用SMO(序列最小优化)算法对6.2.1节的两个式子进行优化: 1.最小化的目标函数 2.优化过程中必须遵循的约束条件

Platt的SMO算法是将大优化问题分解为多个小优化问题来求解的。这些小优化问题往往很容易求解，并且对它们进行顺序求解的结果与将它们作为整体求解的结果是完全一致的，但缩减了许多时间。

原理：每次循环中选择两个alpha进行优化处理，增大其中一个同时减少另一个（因为要满足 alpha\*label的和=0的约束条件）。一对合适的alpha条件: 1. 两个alpha必须要在间隔边界之外 2. 两个alpha还没有进行过区间化处理或者不在边界上

SMO中两个变量的选择过程分别为外循环和内循环，具体见李航《统计学习方法》。

注意：如果修改容错率toler会怎样？改变C的值又如何呢？常数C给出的是不同优化问题的权重。常数C 一方面要保障所有样例的间隔不小于1.0，另一方面又要使分类间隔要尽可能大。如果C很大，那么分类器将力图通过分隔超平面对所有的样例都正确分类，此时得到的支持变量更多。如果数据集非线性可分，就会发现支持向量会在超平面附近聚集成团。得到alpha的值以后，我们可以根据公式计算出权重w。实际上，上面我们计算得到的大部分alpha值为0，而非零alpha所对应的也就是支持向量。因此在计算w的过程中可以将数据集的其他数据点舍弃。计算f(x+b)，若该值大于0，则其属于1类，若该值小于0，则属于-1类。

以上的样本中，两个类的数据点分布在一条直线的两边，但是，倘若两类数据点分别分布在一个圆的内部和外部，那么会得到什么样的分类面呢？使用核函数（kernel）可以将数据转换成易于分类器理解的形式，下面介绍 径向基函数（radial basis function）这一最流行的核函数。径向基函数是一个采用向量作为自变量的函数，能够基于向量距离计算输出一个标量。

利用核函数可以将数据映射到高维空间。可以将核函数当作一个包装器（wrapper）或接口（interface），它能把数据从某个很难处理的形式转换为另一个很容易处理的形式。我们可以在高维空间中解决线性问题，也就等价于在低维空间中解决非线性问题。SVM优化中所有的运算都可以写成内积（inner product,点积）的形式，我们可以把内积运算替换为核函数，而不必做简化处理。把内积替换为核函数的方式被称为核技巧（kernel trick）或者核“变电”(kernel substation)。

有一个参数是用户定义的用于确定到达率（reach）或者说函数值跌落到0的速度参数。参数非常重要，存在一个最优值，直接影响支持向量的数目以及测试错误率。该数据集在这个设置某处存在最优值，如果降低，那么训练错误率就会降低，但是测试错误率却会上升（过拟合？）支持向量的数目存在一个最优值，SVM的优点在于它能对数据进行高效分类。如果支持向量太少，就会得到一个很差的决策边界。若太多，相当于每次都利用整个数据集进行分类，那就变成了k-近邻。SVM训练完成以后只需保留支持向量即可应用。

常数C和参数（如何设置还是个问题，不同特征的大小都会影响参数的取值）都会影响分类的结果。存在另外的SVM形式，它们把C也同时考虑到了优化过程中，例如 v-SVM。注意：最小的训练错误率并不对应于最小的支持向量数目。有时线性核函数的效果并不糟糕，可以以牺牲线性核函数的错误率来换取分类速度的提高。

小结：支持向量机之所以称为‘机’是因为它会产生一个二值决策结果，即他是一种决策‘机’。支持向量机的泛化错误率较低，也就是说它具有良好的学习能力，且学到的结果具有很好的推广性。支持向量机试图通过求解一个二次优化问题来最大化分类间隔。

#### 6.AdaBoost元算法提高分类性能

# 元算法（mean-algorithm）是对其他算法进行组合的一种方式，AdaBoost就是最流行的元算法

# 前面已经介绍了五种不同的分类算法，它们各有优缺点，我们自然可以将不同的分类器组合起来：这种组合结果就是集成方法或者元算法。使用集成方法时会有多种形式：可以是不同算法的集成，也可以是同一算法在不同设置下的集成，还可以是数据集不同部分分配给不同分类器之后的集成。

AdaBoost优点：泛化错误率低，易编码，可以应用在大部分分类器上，无参数调整

AdaBoost缺点：对离群点敏感

bagging（自举汇聚法 bootstrap aggregating）是基于数据随机重抽样的分类器构建方法，是从原始数据集选择S次（放回抽样）后得到S个新数据集的一种技术。在S个数据集建好以后，将某个学习算法分别作用于每个数据集就得到了S个分类器，选择分类器投票结果中最多的类别作为最后的分类结果。随机森林是一种更先进的bagging方法。

与bagging不同的是，boosting通过集中关注被已有分类器错分的那些数据来获得新的分类器。bagging中的分类器权重是相等的，而boosting中的分类器权重并不相等，每个权重代表的是其对应分类器在上一次迭代中的成功度。Boosting算法要涉及到两个部分，加法模型和前向分步算法：加法模型就是说强分类器由一系列弱分类器线性相加而成；前向分步就是说在训练过程中，下一轮迭代产生的分类器是在上一轮的基础上训练得来的 Fm(x)=Fm−1(x)+βm\*hm\*(x;am)。本书采用的单层决策树作为弱分类器，作为弱分类器，简单分类器的效果更好。

由于采用的损失函数不同，Boosting算法也因此有了不同的类型，AdaBoost就是损失函数为指数损失的Boosting算法，AdaBoost是adaptive boosting（自适应boosting）的缩写，其运行过程如下：训练数据中的每个样本，并赋予相等权重，这些权重构成了向量D。首先在训练数据上训练出一个弱分类器并计算该分类器的错误率。在分类器的第二次训练当中，第一次分对的样本的权重会降低，第一次分错的样本的权重会提高。除此之外，Adaboost还为每个分类器都分配了一个权重值alpha，这些alpha值是基于每个弱分类器的错误率进行计算的，错误率=，alpha=，计算出alpha值之后就可以对权重向量D进行更新（正确样本与错误样本的更新公式不同）。计算出D以后，Adaboost又开始进入下一轮迭代（以此时的权重向量D的样本再次创建一个最佳参数的弱分类器，classifierArr是训练出的多个弱分类器，这就是最终的强分类器（由许多弱分类器组成）），知道训练错误率为0或者弱分类器的数目达到用户的指定值为止。

测试算法：遍历classifierArr中的所有弱分类器,将它们输出的类别预测值乘上该单层决策树的alpha权重然后累加到aggClassEst,aggClassEst大于0则返回+1，小于0则返回-1

总结：AdaBoost和SVM是监督学习中最强大的两种方法，我们可以将弱分类器想象成SVM中的一个核函数，也可以按照最大化某个最小间隔的方式重写AdaBoost算法。而他们的不同之处就在于其定义的间隔计算方式有所不同（？），因此导致的结果也不同。

#### 7.预测数值型数据：回归

回归的目的是预测数值型的目标值，最直接的方法是依据输入写出一个目标值得计算公式，这就是所谓的回归方程（regression equation）。回归需要数值型数据，标称型数据将被转为二值型数据。

优点：结果易于理解，计算上不复杂

缺点：对非线性的数据拟合不好

采用OLS，即普通最小二乘法（ordinary least squares）求解w时，需要对矩阵求逆，因此P138推导出的方程只有在逆矩阵存在的时候适用。必须在代码中对矩阵的逆是否存在做出判断。linalg是numpy中线性代数的库，det用于计算行列式。

最佳拟合直线方法将数据视为直线进行建模，几乎任一数据集都可以用这个方法来建立模型，那么如何判断这些模型的好坏呢？可以通过计算两个序列的相关系数来计算预测值yHat序列和真实值y序列的匹配程度

yHat=xMat\*ws  
print corrcoef(yHat.T,yMat)   
# 左：预测值，右：实际值，需要将yMat转置，以保证两个向量都是行向量  
# 对角线上数据为1.0，因为和自己百分百相关

线性回归求的是具有最小均方误差的无偏估计，所以有可能出现欠拟合现象，因此有些方法允许在估计中引入一些偏差，从而降低预测的均方误差,其中的一个方法是采用局部加权线性回归函数（locally weighted linear regression，LWLR）。在该算法中，我们给待预测点（即测试样本点）附近的每个点赋予一定的权重，然后在这个子集上基于最小均方差来进行普通的回归。这种算法每次预测均需要事先选取出对应的数据子集，该算法解出回归系数w的形式见P142。LWLR使用“核”（与SVM中的核类似）来对附近的点赋予更高的权重，最常用的核是高斯核，随着距离接近，权重以指数级衰减，参数k决定了对附近的点赋予多大的权重。书中给的例子，k=0.5时，大部分数据都被用于训练回归模型，k=0.01时，仅有很少的局部点被用于训练回归模型。

书中给出的例子显示，简单线性回归达到了与局部加权线性回归类似的效果。这也表明一点，必须在未知数据上比较效果才能选取到最佳模型。

局部加权线性回归存在一个问题，即增加了计算量，因为它对每个点做预测时都必须使用整个数据集。K=0.01时可以得到很好的估计，但可以发现此时大多数数据点的权重都接近于0。如果避免这些计算将可以减少程序运行时间，从而缓解因计算量增加带来的问题。下面将介绍另一种提高预测精度的方法，并分析它的优势所在。

# 缩减系数来“理解”数据

# 如果数据的特征比样本点还多，则输入数据的矩阵不是满秩矩阵，所以X(T)X 无法求逆，就无法使用线性回归和原来的方法来做预测

# 为了解决这个问题，可使用岭回归(ridge regression)、lasso法(效果很好但计算复杂)、前向逐步回归(效果不错且容易实现)

**岭回归**：就是在矩阵X(T)X上加一个I从而使得矩阵非奇异，进而能求逆（和正则化参数差不多，越大，系数的减少就更大）。岭回归最初用来处理特征数多于样本数的情况，现在也用于在估计中加入偏差，从而得到更好的估计。通过引入来限制了所有的w之和，通过引入该惩罚项，能够减少不重要的参数（从而更好的理解数据），这个技术在统计学中叫做缩减（shrinkage）。通过选取不同的来重复上述过程，最终得到一个使预测误差最小的。注意，岭回归需要对数据进行标准化处理，标准回归则没有。

做出鲍鱼上8个特征值的权重改变图。最小时，可以得到所有系数的原始值（与线性回归一致）；而在右边，系数全部缩减为0；在中间部分的某值将可以取得最好的预测效果。

ridgeWeights=ridgeTest(abX,abY)  
ax.plot(ridgeWeights)   
# 横坐标为1-30，代表矩阵中的30行（lambda的值逐渐增大），纵坐标为8个特征值的权重改变线图（是矩阵的每一行上面的值）  
a=ridgeWeights  
plt.show()

若限定所有回归系数的平方和不能大于，增加此约束之后，普通的最小二乘法回归会得到与岭回归一样的公式。使用普通的最小二乘法回归在当两个或更多的特征相关时，可能会得出一个很大的正系数和一个很大的负系数。正是因为上述限制条件的存在，使用岭回归可以避免这个问题。

**lasso：**与岭回归类似，另一个缩减方法lasso也对回归系数做了限定。不同点在于此处限定所有回归系数的绝对值之和不能大于。此时，在足够小的时候，一些系数会因此被迫缩减到0，这个特性可以帮助我们更好地理解数据。为了在这个新的约束条件下解出回归系数，需要使用二次规划算法，极大地增大了计算复杂度。

**前向逐步回归：**前向逐步回归算法属于一种贪心算法，即每一步都尽可能减少误差。一开始，所有的权重都设为1，然后每一步所做的决策是对某个权重增加或减少一个很小的值。

伪代码如下： 在每轮迭代过程中：

设置当前最小误差lowestError为正无穷

对每个特征增大或缩小：

改变一个系数得到一个新的w，计算新w下的误差

如果误差error小于当前最小误差：设置Wbest等于当前的w

将w设置为新的wbest

lowestError=inf # 设置当前最小误差为无穷  
for j in range(n): # 对所有特征进行循环  
 for sign in [-1,1]: # 分别计算增加或减少该特征对误差的影响  
 wsTest=ws.copy()  
 wsTest[j] += eps\*sign

# eps表示每次迭代需要调整的步长

# 可以观察到w1、w6都是0，这表明它们不对目标值造成任何影响，也就是这些特征很可能是不需要的

# 另外，在参数eps设置为0.01的情况下，一段时间后系数就已经饱和了，并在0.04和0.05之间来回震荡，这是因为步长太大的缘故

# 逐步线性回归算法的优点在于它可以帮助人们理解现有的模型并做出改进，当构建了一个模型后，可以运行算法找出重要的特征，并停止对不重要特征的收集

# 如果用于测试，该算法可以用来选择使误差最小的模型

# 当应用缩减方法(如逐步线性回归或岭回归)时，模型增加了偏差（bias），与此同时也减小了模型的方差

小结：数据集上计算出的回归过程并不一定意味着它是最佳的，可以通过预测值yHat和原始值y的相关性来度量回归方程的好坏。

当数据样本数比特征数还少的时候，矩阵的逆不能直接计算。即使当样本数比特征数多时，如果特征存在高度相关，矩阵的逆也无法计算。这时可以考虑使用岭回归，因为当逆无法计算时，岭回归仍能保证求得回归参数。也可采用lasso以及简便的逐步线性回归方法。

缩减法还可以看做是对一个模型增加偏差的同时减少方差。偏差方差折中是一个重要的概念，可以 帮助我们理解现有模型并做出改进，从而得到更好的模型。

#### 8.树回归（CART算法）

前面介绍的线性回归创建的模型需要拟合所有的样本（局部加权线性回归除外）。当数据拥有众多特征且特征之间关系十分复杂时，构建全局模型就太难了。并且，实际生活中很多问题都是非线性的，不可能使用全局线性模型来拟合任何数据。

一种可行的方法是将数据集切分成很多份易建模的数据，然后用线性回归技术来建模，如果首次切分后仍然难以拟合线性模型就继续切分。采用CART（classification and regression trees，分类回归树）的树构建算法。该算法既可以用于分类还可以用于回归，因此非常值得学习。回归树和分类树的思路类似，但叶节点的数据类型不是离散型，而是连续型。树回归需要数值型的数据，标称型数据应该映射成二值型数据。

优点：可以对复杂和非线性的数据建模

缺点：结果不易理解

前面学习过的ID3算法不能直接处理连续型特征，只有事先将连续型特征转换为离散型，才能使用，但这种转换过程会破坏连续型变量的内在性质。ID3按照某特征的所有可能值来切分，可以采用二元切分法，即每次把数据集切成两份，此方法易于处理连续型特征，具体处理方法：如果特征值大于给定值就走左子树，否则就走右子树。

要对数据的复杂关系建模，我们已经决定借用树结构来帮助切分数据。那么如何实现数据的切分呢？怎么才能知道是否已经充分切分呢？这些问题的答案取决于叶节点的建模方式。回归树假设叶节点是常数值，这种策略认为数据中的复杂关系可以用树结构来概括。为成功构建以分段常数为叶节点的树，需要度量出数据的一致性，之前ID3使用树进行分类时，会在给定节点计算数据的无序度。那么**如何计算连续型数值的混乱度**呢：首先计算所有数据的均值，然后计算每条数据的值到均值的差值。一般用绝对值或平方值来代替上述差值，平方误差的总值（总方差）可以通过均方差乘以数据集中样本点的个数 得到。

给定某个误差计算方法，该函数会找到数据集上最佳的二元切分方式，该函数还要确定什么时候停止切分，一旦停止切分会生成一个叶节点。如果切分数据集后效果提升不够大，那么就不应进行切分操作而直接创建叶节点。

if (S-bestS) < tolS:   
 # 如果切分数据集后效果提升不够大，那么就不应进行切分操作而直接创建叶节点  
 return None,leafType(dataSet)

if (shape(mat0)[0] < tolN) or (shape(mat1)[0] < tolN):   
 # 如果切分出的数据集很小则退出  
 return None,leafType(dataSet)

一棵树如果节点过多，表明该模型可能对数据进行了“过拟合”，通过降低决策树的复杂度来避免过拟合的过程称为剪枝（pruning）

在函数chooseBestSplit()中的提前终止条件，实际上就是所谓预剪枝（prepruning）操作

另一种形式的剪枝需要使用测试集和训练集，称为后剪枝（postpruning），本节将分析后剪枝的有效性

# 预剪枝

# 树构建算法对输入参数tolS和tolN非常敏感，如果使用其他值，构建的效果可能会大打折扣

# 停止条件tolS对误差的数量级十分敏感，然而，通过不断修改停止条件来得到合理结果并不是很好的方法

# 后剪枝

# 使用后剪枝方法需要将数据集分成测试集和训练集

# 首先指定参数，使得构建出的树足够复杂，便于剪枝

# 接下来从上而下找到叶节点（如果存在任一子集是一棵树，则在该子集递归剪枝过程），用测试集来判断这些叶节点合并是否能降低测试误差

# 模型树（将叶节点设定为分段线性函数)

# 分段线性(piecewise linear)是指模型由多个线性片段组成：数据集里的一部分数据以某个线性模型建模，而另一部分数据则以另一个线性模型建模，这就是所谓的分段线性模型。很显然，两条直线比很多节点组成一棵大树更容易解释。模型树的可解释性是它优于回归树的特点之一。另外，模型树也具有更高的预测准确度。

对于模型树来说，为了找到最佳切分，应该怎样计算误差呢？前面用于回归树的误差计算方法这里不能再用。稍加变化，对于给定的数据集，应该先用线性的模型来对它进行拟合，然后计算真实的目标值与模型预测值间的差值。最后将这些差值的平方求和就得到了所需的误差。（这不是和线性回归计算误差的方法一样的嘛）

# 思考一个问题：模型树、回归树以及第8章里的其他模型，哪一种模型更好呢？一个比较客观的方法是计算相关系数

# 该相关系数可以通过调用numpy库中的命令corrcoef(yHat,y,rowvar=0)来求解

# 结果显示，对书中提供的数据：模型树>回归树>标准线性回归

#### 9.K-均值聚类对未标注数据分组

聚类即无监督分类，它试图将相似对象归入同一簇，将不相似对象归到不同簇。K均值（K-means）算法之所以叫这个名字是因为它可以发现k个不同的簇，且每个簇的中心采用簇中所含值得均值计算而成。

优点：容易实现

缺点：可能收敛到局部最小值，在大规模数据集上收敛较慢

适合数据类型：数值型数据

工作流程：

创建k个点作为起始质心（经常是随机选择），将数据集中的每个点分配到一个簇中，具体来讲，为每个点找距其最近的质心，将其分配给该质心所对应的簇。这一步完成之后，每个簇的质心更新为该簇所有点的平均值。此时对于每个点再重新寻找距其最近的质心，重新分配簇。不断重复此过程，直到数据点的簇分配结果不再发生改变（或达到指定的循环次数）。

# 使用后处理来提高聚类性能

# 如何判断簇数目的选择是最合理的呢？在包含簇分配结果的矩阵中保存着每个点的误差，即该点到簇质心的距离平方值。

# K-均值算法收敛但聚类效果较差的原因是，K-均值算法收敛到了局部最小值，而非全局最小值

# 利用SEE(sum of squared error,误差平方和)度量聚类效果，即clusterAssment矩阵的第一列之和

# 因为对误差取了平方，因此更加重视那些远离中心的点。注意，**聚类的目标是在保持簇数目不变的情况下提高簇的质量**

# 方法1：将具有最大SEE值的簇划分为两个簇，将最大簇包含的点过滤出来并在这些点上运行k=2的k-均值算法

# 为了保证簇总数不变，可以将某两个簇进行合并（2 ways）

# 二分K-均值算法

# 该算法首先将所有点作为一个簇，然后将该簇一份为二，之后选择一个簇继续划分，选择哪个簇取决于对其划分是否可以最大程度降低SEE的值

# 另一种做法是选择SEE最大的簇进行划分，知道簇数目达到用户指定的数目为止

#### 10.Apriori算法进行关联分析

从大规模数据集中寻找物品间的隐含关系被称作关联分析（association analysis）或者关联规则学习（association rule learning）。关联分析的目标包含两项：发现频繁项集和关联规则。频繁项集（frequent item sets）是经常出现在一块的物品的集合，关联规则（association rules）暗示两种物品之间可能存在很强的关系。首先需要找到频繁项集，然后才能获得关联规则。

Apriori 意为先验的，演绎的，推测的。

优点：易编码实现

缺点：在大数据集上可能较慢

一个项集的支持度（support）被定义为数据集中包含该项集的记录所占的比例，如5条交易记录中有三条包含{豆奶，尿布}，因此它的支持度为3/5。支持度是针对项集来说的，因此可以定义一个最小支持度，而只保留满足最小支持度的项集。

可信度（confidence）是针对一条诸如{尿布}——{葡萄酒}的关联规则来定义的。这条规则的可信度被定义为 支持度{尿布，葡萄酒}/支持度{尿布}。

假设我们在经营一家杂货店，对那些经常在一起购买的商品非常感兴趣。对于包含N种物品的数据集共有2的N次方-1种项集组合，要得到这些组合的支持度显然太复杂了。可利用Apriori原理：如果某个项集是频繁的，那么它的所有子集也是频繁的，意味着若{0，1}是频繁的，那么{0}，{1}也一定是频繁的。也就是说如果一个项集是非频繁的，那么它的所有超集也是非频繁的。一旦计算出{2，3}的支持度，知道{2，3}是非频繁的之后，就不需要再计算{0，2，3}，{1，2，3}，{0，1，2，3}的支持度了。使用该原理就可以避免项集数目的指数增长。

#利用Apriori算法来发现频繁集

Apriori算法的两个输入参数分别是最小支持度和数据集。该算法首先会生成所有单个物品的项集列表，接着扫描交易记录来查看哪些项集满足最小支持度要求，那些不满足最小支持度的集合会被去掉。然后，对剩下的集合进行组合以生成包含两个元素的项集。接下来重新扫描交易记录去掉不满足最小支持度的项集。重复此过程。

具体说来就是：首先创建一个空列表，用来存储所有不重复的项值。遍历数据集中的所有交易记录，对每一条记录，遍历记录中的每一个项。如果某个物品项没有在C1中出现，则添加到C1中。这里并不是简单地添加每个物品项，而是添加只包含该物品项的一个列表（也就是说，C1是一个集合的集合，每次添加的都是单个项构成的集合 {{1}，{2}，{4}……}）。

整个Apriori算法的伪代码如下：

当集合中项的个数大于0时：

构建一个k个项组成的候选项集的列表

检查数据以确认每个项集都是频繁的

保留频繁项集并构建k+1项组成的候选项集的列表

举例来说，该函数以{0}，{1}，{2}作为输入，会生成{0，1}，{0，2}，{1，2}。接下来，比较频繁项集Lk中的每一个元素与其他元素，这可以通过两个for循环来实现。紧接着，取列表中的两个集合进行操作，若前k-2个元素都相等，那么进行合并。

之前给出了频繁项集的量化定义，即它满足最小支持度要求。对于关联规则，我们也有类似的量化方法，这种量化指标称为可信度。一条规则P——>H的可信度定义为support(P|H)/support(P)，| 表示集合的并操作。前一节已经计算了所有频繁项集支持度，现在想获得可信度，所需要做的只是取出那些支持度值做一次除法运算。类似于上一节的频繁项集生成，我们可以为每个频繁项集产生许多关联规则。可以观察到，如果某条规则并不满足最小可信度要求，那么该规则的所有子集也不会满足最小可信度要求。假设规则 0，1，2——>3不满足最小可信度要求，那么就知道任何左部为{0,1,2}子集的规则也不会满足最小可信度要求。如{0，1} ——>任何一个集合 都不会满足最小可信度要求。

可以利用关联规则的上述性质来减少需要测试的规则数目：首先从一个频繁项集开始，接着创建一个规则列表，其中规则右部只包含一个元素，然后对这些规则进行测试。接下来合并所有剩余规则来创建一个新的规则列表，其中规则右部包含两个元素。这种方法也被称为分级法（没看懂）。函数最后要生成一个包含可信度的规则列表（生成列表的多少以及项集的复杂程度取决于最小可信度minConf）

总结：每次增加频繁项集的大小，Apriori算法都会重新扫描整个数据集。当数据集很大时，这会显著降低频繁项集发现的速度。下一章介绍的FP-growth算法只对数据库进行两次遍历，能够显著加快发现频繁项集的速度。

#### 11.FP-growth算法来高效发现频繁项集

搜索引擎公司的工程师们通过查看互联网上的用词来找出经常在一起出现的词对，这需要一种高效发现频繁集的方法。可以利用FP-growth算法，它基于Apriori构建，但比先验算法更快。FP-growth将数据集存储在一个特定的称作FP树的结构之后，再去发现频繁项集或者频繁项对，即常在一块出现的元素项的集合FP树。前一章介绍了发现频繁项集与关键规则的算法，FP-growth能更为高效地发现频繁项集，但不能用于发现关联规则。

Apriori算法对每个潜在的频繁项集都会扫描数据集判定给定模式是否频繁，而FP-growth算法只会扫描数据集两次，它发现频繁项集的基本过程如下：

1. 构建FP树
2. 从FP树中挖掘频繁项集

优点：一般要快于Apriori

缺点：实现比较困难，在某些数据集上性能会下降

适用数据类型：标称性数据

FP-growth算法将数据存储在一种称为FP树的紧凑数据结构中，FP代表频繁模式（Frequent Pattern），一棵FP树通过链接（link）来连接相似元素，被连起来的元素项可以看成一个链表。同搜索树不同的是，一个元素项可以在一棵FP树中出现多次。FP树会存储项集的出现频率，而每个项集会以路径的方式存储在树中。

FP-growth算法的工作流程如下。首先构建FP树，然后利用它来挖掘频繁项集。为构建FP树，需要对原始数据集扫描两遍。第一遍对所有元素项的出现次数进行计数。记住，如果某元素是不频繁的，那么包含该元素的超集也是不频繁的，所以就不需要考虑这些超集。数据库的第一遍扫描用来统计出现的频率，而第二遍扫描中只考虑那些频繁元素（第二次扫描数据集时会构建一棵FP树，本章的FP树要比书中其他树更加复杂，因此要创建一个类来保存树的每一个节点）。

注意FP-growth算法中存储的是集合，所以需要离散数据，如果要处理连续数据，需要将它们量化为离散值。

需要一个头指针来指向类型的第一个实例，利用头指针，可以快速访问FP树中一个给定类型的所有元素。这里使用一个字典作为数据结构，来保存头指针表。除了存放指针外，头指针表还可以用来保存FP树中每类元素的总数。

第一次遍历数据集会获得每个元素项的出现频率。接下来，去掉不满足最小支持度的元素项。再下一步构建FP树。在构建时，读入每个项集并将其添加到一条已经存在的路径中。如果该路径不存在，则创建一条新路径。在将集合添加到树之前，需要对每个集合进行排序。排序基于元素项的绝对出现频率来进行。

在对事务记录过滤和排序之后，就可以构建FP树了。从空集开始，向其中不断添加频繁项集。过滤、排序后的事务依次添加到树中，如果树中已存在现有元素，则增加现有元素的值；如果现有元素不存在，则向树添加一个分枝。

构建好FP树之后，就可以抽取频繁项集了。首先从单元素项集合开始，然后在此基础上逐步构建更大的集合。当然这里将利用FP树来实现上述过程，不再需要原始数据集了。从FP树种抽取频繁项集的三个基本步骤如下：

1. 从FP树中获得条件模式基（conditonal pattern base）
2. 利用条件模式基，构建一个条件FP树
3. 迭代重复步骤(1)步骤(2)，直到树包含一个元素项为止。

条件模式基是以所查找元素项为**结尾**的路径集合。每一条路径其实都是一条前缀路径（prefix path）。简而言之，一条前缀路径是介于所查找元素项与树根节点之间的所有内容。为了获得这些前缀路径，可以对树进行穷举式搜索，直到获得想要的频繁项为止。另一种更有效的方法是利用先前创建的头指针。头指针表包含相同类型元素链表的起始指针。一旦到达了每一个元素项，就可以上溯这棵树直到根节点为止。

对于每一个频繁项，都要创建一棵条件FP树。详情见P233

FP-growth算法最终得到的结果是如P234所示的频繁项。

### 算法实例

#### 1.k-近邻算法改进约会网站的配对效果p20

将交往过的三种类型的男子按照特征来正确分类：不喜欢，一般般，有魅力



1. 准备数据：从文本文件中解析数据
2. 分析数据：使用Matplotlib创建散点图
3. 准备数据：归一化数值
4. 测试算法：作为完整程序验证分类器
5. 使用算法：构建完整可用系统

#### 2.k-近邻算法的手写识别系统p28

1. 准备数据：将图像转换为测试向量
2. 测试算法：识别手写数字

#### 3.决策树预测隐形眼镜类型p50

1. 准备数据：解析tab键分隔的数据行
2. 分析数据：使用createPlot()函数绘制最终的树形图
3. 训练算法：使用3.1节的createTree()函数
4. 测试算法
5. 使用算法：存储树的数据结构（使用pickle模块）

#### 4.朴素贝叶斯进行文本分类p57

1. 准备数据：从文本中构建词向量
2. 训练算法：从词向量计算概率
3. 测试算法：根据现实情况修改分类器（解决概率为0与下溢出问题）
4. 准备数据：文档词袋模型

#### 5.朴素贝叶斯过滤垃圾邮件p65

1. 准备数据：切分文本
2. 测试算法：使用朴素贝叶斯进行交叉验证

#### 6.朴素贝叶斯从个人广告中获取区域倾向p68

# 应用：基于一个人的用词来推测他的年龄和其他信息

# 我们将分析两个不同城市的人们发布的征婚广告信息，来比较这两个城市的人们在广告用词上是否不同。如果确实不同，那么他们各自常用的词是哪些？

# 从人们的用词当中，我们能否对不同城市的人所关心的内容有所了解？

1. 收集数据：导入RSS源（并去除高频词）
2. 分析数据：显示地域相关的用词

#### 7.逻辑回归从疝气病症预测病马的死亡率p85

1. 准备数据：处理数据中的缺失值

本例中所有缺失值必须用一个实物值来替换，因为numpy数据类型不允许包含缺失值。这里选择0来替换所有缺失值，在逻辑回归的权重更新公式中，如果某特征对应值为0，那么该特征的系数将不做更新。并且由于sigmoid(0)=0.5，它对结果的预测不具有任何倾向性。若逻辑回归中的类别标签缺失，直接丢弃这个样本就好。

1. 测试算法：用逻辑回归进行分类

#### 8.支持向量机进行手写文字的识别p111

#### 9.在一个难数据集上应用AdaBoost p125

在之前给出的马疝病数据集上应用AdaBoost分类器。实验结果显示：由于有30%的数据缺失，所以效果不好，但也比逻辑回归好。本实验中改变分类器数目，测试错误率在达到了一个最小值之后有开始上升了，过拟合了呀。有文献表明，对于表现好的数据集，AdaBoost的测试错误率会到达一个稳定值（我们的数据集不够好）

#### 10.线性回归预测鲍鱼的年龄P145

可以看到，使用较小的核将得到较低的训练误差。但使用最小的核可能会造成过拟合

它的测试误差并非是最小的，核=10的测试误差反而最小（虽然它的训练误差最大）

见P152.可见，随着核逐渐减小，训练误差将变小，偏差逐渐减小（**偏差**是模型预测值和数据之间的差异），测试误差（方差）先变小后变大（过拟合）。

**反差**是模型之间的差异，方差是可以度量的，分别取出两组随机样本集并分别用线性模型拟合得到两组回归系数，这些系数间的差异大小也就是模型方差大小的反映。因此，随着模型复杂度的提高，方差会变大。

注：这里的偏差和方差和 吴恩达 讲的是同一个概念。偏差大：欠拟合。方差大：欠拟合或过拟合

那么，最佳的核大小是10吗？maybe，如果想得到更好的效果，应该用10个不同的样本集

做10次测试来比较结果。

简单线性回归达到了与局部加权线性回归类似的效果。这也表明一点，必须在未知数据上比较效果才能选取到最佳模型。

#### 11.用回归法预测乐高套装的价格P153

1.收集数据：使用Google购物的API

发送HTTP请求后API将以JSON格式返回所需的产品信息，python提供了JSON解析模块，我们可以从返回的JSON格式里整理出所需数据

2.训练算法：建立模型

lgX1[:,1:5]=mat(lgX)   
# 添加常数项特征X0（X0=1）

ws=standRegress(lgX1,lgY)

从调用standRegress函数得到的公式可以看出：套装里零部件越多售价反而会越低，很明显这是个错误的结论。因此虽然它对数据拟合得很好，预测效果也不错，但模型本身并不能令人满意。下面使用岭回归再进行一次实验，并演示如何用缩减法确定最佳回归系数。

random.shuffle(indexList)   
# 对indexList进行混洗（shuffle），从而实现数据集数据点的随机选取

# 注意岭回归使用了数据标准化，standRegres则没有，为了将上述比较可视化还需将数据还原

matTestX = (matTestX-meanTrain)/varTrain   
# 注意要用训练样本对测试样本做归一化  
yEst = matTestX \* mat(wMat[k,:]).T + mean(trainY)   
# 注意得到预测的数据需要加上mean(trainY)

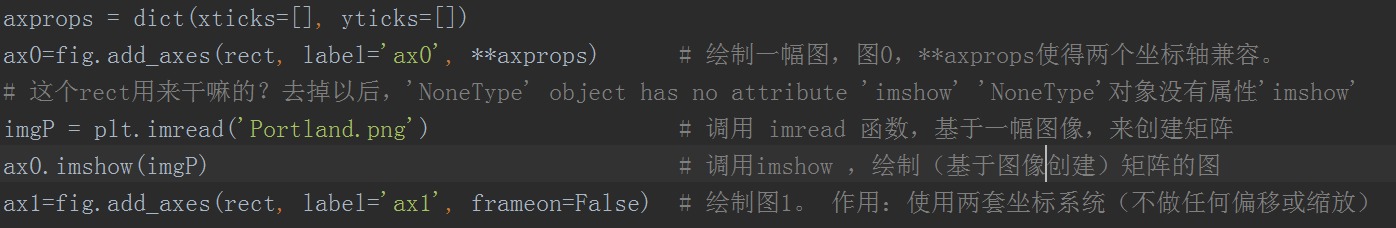
unReg = bestWeights/varX   
# 权重还原的公式，除以xMat的偏差即可，注意需要简单证明

将此时得到的公式与常规的最小二乘法得到的公式进行比较，结果显示结果并没有多大差异，我们本期望找到一个更易于理解的模型，显然没有达到预期效果。我们来看一下在缩减过程中回归系数是如何变化的。

输入：ridgeTest(lgX,lgY) ，结果显示，这些系数是经过不同程度的缩减得到的，看第一行，第四项比第二项的系数大5倍，第二项比第一项大57倍。如果只能选择一个特征来做预测的话，我们应该选择第四个特征，或第四以及第二个特征。这种分析方法十分有效，**它可以指出哪些特征是关键的，而哪些特征是不重要的**。

#### 12.对地图上的点进行聚类

在一张图片上运行聚类的结果，并且绘制兼容的坐标轴



注意！使用索引i % len(scatterMarkers) 来选择标记形状，这样就可以循环地使用这些标记，无论要求的簇数目是多少，利用整除取余，总能循环

for i in range(numClust):  
 ptsInCurrCluster = datMat[nonzero(clustAssing[:,0].A==i)[0],:]  
 markerStyle = scatterMarkers[i % len(scatterMarkers)]  
 # 注意！使用索引i % len(scatterMarkers) 来选择标记形状，这样就可以循环地使用这些标记  
 ax1.scatter(ptsInCurrCluster[:,0].flatten().A[0], ptsInCurrCluster[:,1].flatten().A[0],\  
 marker=markerStyle, s=90)

#### 13.Apriori算法发现国会投票中的模式P212

需要使用votesmart API

#### 14.Apriori算法发现毒蘑菇的相似特征P220

有时我们并不想寻找所有的频繁项集，而只对包含某个特定元素项的项集感兴趣。此例中，我们会寻找毒蘑菇中的一些公共特征，利用这些特征就能避免吃到那些有毒的蘑菇。有一个关于蘑菇的23种特征的数据集，第一个特征表示有毒或者可食用，如果某样本有毒，则值为2。为了找到毒蘑菇中存在的公共特征，可以运行Apriori算法来寻找特征值为2的频繁项集。

mushDatSet = [line.split() for line in open('mushroom.dat').readlines()]  
L,suppData = apriori(mushDatSet,minSupport=0.3)  
# 在结果中搜索包含有毒特征值2的频繁项集  
for item in L[1]: # L[1]由2个项集组合  
 if item.intersection('2'):print item,'与2同时出现的单个特征'   
# 也可以对更大的项集来重复上述过程  
for item in L[3]: # L[1]由4个项集组合  
 if item.intersection('2'):print item,'与2同时出现3个特征'

#### 15.FP-growth在twitter源中发现一些共现词P235

1．收集数据：导入twitter库及正则表达式re模块

2．准备数据：编写一个函数来去掉URL，去掉标点，转换为小写并从字符串中建立一个单词集合

我们之前是直接遍历所有的数据来构建FP树的，实际上可以重写createTree()函数，每次读入一个实例，并随着Twitter流的不断输入而不断增长树。FP-growth算法还有一个map-reduce版本的实现。

#### 16. FP-growth从 新闻网站点击流 中挖掘（大数据）P238

parsedDat = [line.split() for line in open('kosarak.dat').readlines()]   
# 将数据集导入到列表  
initSet = createInitSet(parsedDat) # 对初始集合格式化  
myFPtree,myHeaderTab = createTree(initSet,100000)   
# 构建FP树，并从中寻找那些至少被10万人浏览过的新闻报道  
myFreqList = [] # 创建一个空列表来保存这些频繁项集  
mineTree(myFPtree,myHeaderTab,100000,set([]),myFreqList)  
print len(myFreqList)   
# 看看有多少新闻报道或报道集合曾经被10万或者更多的人浏览过  
print myFreqList # 看看具体有哪些新闻

### 3.常用代码

#### 1.小技巧

shape[0]返回行数，shape[1]返回列数

diffMat=tile(inx,(dataSetSize,1))-dataSet

tile函数用于将inx扩展为 训练样本的个数dataSetSize（某常数） \*inx行数，1\*inx原来的列数 的矩阵

sqDistances=sqDiffMat.sum(axis=1)

# axis＝0表示按列相加，axis＝1表示按照行的方向相加，得到一个列向量

补充：dataSet.min(0) 0:取列的最小值，1：取行的最小值

sortedDistIndicies=distances.argsort()

# argsort函数返回数组值从小到大的索引值

classCount[voteIlabel]=classCount.get(voteIlabel,0)+1 # 记录出现次数的好方法啊

voteIlabel是k个最小的索引值对应的标签，classCount={}，是一个空的字典

dict.get(key, default=None)

* key -- 字典中要查找的键。
* default -- 如果指定键的值不存在时，返回该默认值值。

写一个for循环，首先将voteIlabel中所有标签值作为key存入字典中，每当这个标签值加1，value的值就+1，这样就记录下了所有标签号以及它们出现的次数

sortedClassCount=sorted(classCount.iteritems(),key=operator.itemgetter(1),reverse=True)

sorted是个排序函数，接收3个参数，一个可迭代的对象；一个回调函数（回调函数只能有一个参数（如果有多个参数，请用偏函数），根据这个函数的返回值进行排序；一个布尔值，默认为False 升序排列，True即此处按降序排列，那么sortedClassCount[0][0]对应的标签就是出现个数最多的。

**字典的items方法作用：是可以将字典中的所有项，以列表方式返回。因为字典是无序的，所以用items方法返回字典的所有项，也是没有顺序的。**

**字典的iteritems方法作用：与items方法相比作用大致相同，只是它的返回值不是列表，而是一个迭代器。**

**（1）在Python2.x中，iteritems() 用于返回本身字典列表操作后的迭代器【Returns an iterator on all items(key/value pairs) 】，不占用额外的内存。**

**（2）在Python 3.x 里面，iteritems()方法已经废除了。在3.x里用 items()替换iteritems() ，可以用于 for 来循环遍历。**

# operator.itemgetter(1)表示获取对象的第2个域的值，即字典classCount的value值，即各个标签对应的个数

# operator.itemgetter(1,0)则表示获取对象的第2个域和第1个的值，注意返回的key是一个函数，通过key(对象)的方式才能获取具体的值

readlines()自动将文件内容分析成一个行的列表

readline每次只读一行，用于循环读出文件的前n行

append：使用append的时候，是将new\_media看作一个对象，整体打包添加到music\_media对象中

['compact disc', '8-track tape', 'long playing record', ['DVD Audio disc', 'Super Audio CD']]

Extend：使用extend的时候，是将new\_media看作一个序列，将这个序列和music\_media序列合并，并放在其后面。

['compact disc', '8-track tape', 'long playing record', 'DVD Audio disc', 'Super Audio CD']

python语言不用考虑内存分配问题，在函数中传递的是列表的引用，在函数内部对列表对象的修改，将会影响该列表对象的整个生存周期。为了消除这个不良影响，我们需要在函数的开始声明一个新列表对象。

python中列表是按照引用方式传递的，为了保证每次调用函数createTree()时不改变原始列表的内容,复制类标签将其存储在新列表变量中

利用 | 求两个集合的并集，

for document in dataSet 每次取出dataSet中的一个list（dataSet由许多个list组成）

def createVocabList(dataSet):   
 # 此函数创建一个包含在所有文档中出现的不重复词的列表  
 vocabSet=set([])  
 for document in dataSet:  
 temp=document  
 vocabSet=vocabSet | set(document)   
 # 求两个集合的并集，不断将文档中出现的新词加入vocabSet中  
 return list(vocabSet)

记录下一个文档中在给定词集合中出现过的次数，注意 创建一个其中元素都为0的向量采用的是 [0] \* len(vocabList) ，zeros((1, len(vocabList))) 也是可以的

def bagOfWords2VecMN(vocabList, inputSet):  
 returnVec = [0] \* len(vocabList)  
 # 创建一个其中元素都为0的向量  
 for word in inputSet:  
 if word in vocabList:  
 returnVec[vocabList.index(word)] += 1  
 return returnVec

通过xxxmat.getA()将matrix形式转换为array,或者使用np.array(xxxarray)函数

同理有np.matrix(xxxarray)函数将array转换为matrix，或使用 mat(s[])，同理也可将list转换为matrix或array。xxx. tolist() 将array或matrix转换为list

sign函数：当x>0，sign(x)=1;

当x=0，sign(x)=0;

当x<0， sign(x)=-1

var函数，var(xMat,0) # axis可以取0或1，取0求样本方差的无偏估计值（除以N-1；对应取1求得的是方差（除以N）

set(dataSet[:, featIndex].T.A.tolist()[0])

set不能对matrix操作，要先转为array

注意！使用索引i % len(scatterMarkers) 来选择标记形状，这样就可以循环地使用这些标记，无论要求的簇数目是多少，利用整除取余，总能循环

for i in range(numClust):  
 ptsInCurrCluster = datMat[nonzero(clustAssing[:,0].A==i)[0],:]  
 markerStyle = scatterMarkers[i % len(scatterMarkers)]  
 # 注意！使用索引i % len(scatterMarkers) 来选择标记形状，这样就可以循环地使用这些标记  
 ax1.scatter(ptsInCurrCluster[:,0].flatten().A[0], ptsInCurrCluster[:,1].flatten().A[0],\  
 marker=markerStyle, s=90)

#### 2.文本记录的读取

def file2matrix(filename):  
 fr=open(filename)  
 arrayOLines=fr.readlines()

# readlines()自动将文件内容分析成一个行的列表  
 numberOfLines=len(arrayOLines) # 得到文件行数  
 classLabelVector=[]  
 index=0  
 for line in arrayOLines: # 循环，逐行进行操作  
 line=line.strip() # 移除字符串头尾某字符，默认为空格或换行符(回车字符)，这样就得到了一整行数据  
 # 40920 8.326976 0.953952   
 listFromLine=line.split('\t') # split通过指定分隔符对字符串进行切片，如果参数 num 有指定值，则仅做num次分割，分为num+1次  
 # ['40920', '8.326976', '0.953952', '3']

这样就将整行数据分割成了一个元素列表，再将元素存入矩阵

returnMat[index,:]=listFromLine[0:3] # 选取前3个元素，存储到特征矩阵中的第 index 行中

lassLabelVector.append(int(listFromLine[-1])) # 最后一个元素是约会的优先级，存储到这个向量中

index +=1 # 行数+1

当样本集中含有多个以独立文件存在的样本时，可以用listdir取得文件的数量

使用k-近邻算法识别手写数字  
from os import listdir  
def handwritingClassTest():  
 hwLabels=[]  
 trainingFileList=listdir('trainingDigits')  
 # 利用listdir返回指定的文件夹包含的文件或文件夹的名字的列表  
 m= len(trainingFileList) # m为样本数量  
 trainingMat=zeros((m,1024))

# trainingMat每一行存储一个样本  
 for i in range(m): # 取出一个样本  
 fileNameStr=trainingFileList[i]

# '0\_113.txt'  
 fileStr=fileNameStr.split('.')[0]  
 # split通过指定分隔符对字符串进行切片,分为0\_013和txt，取[0]得到文件名0\_013，代表这是数字0的第13个样本  
 classNumStr=int(fileStr.split('\_')[0])  
 # 取得第一个字符0，即此样本标签为0  
 hwLabels.append(classNumStr)  
 trainingMat[i,:]=img2vector('trainingDigits/%s' % fileNameStr)   
 # 将图像矩阵转换为向量，便于计算距离

lineArr=line.strip().split()

# strip只能删除开头或是结尾的字符或是字符串。不能删除中间的字符或是字符串

# ()里无参数时，默认删除空白符（包括'\n', '\r', '\t', ' ')

#### 3.归一化数据

下面的函数可以自动将数字特征值转化为0到1的区间

公式：new=(old-min)/(max-min)

def autoNorm(dataSet):  
 minVals=dataSet.min(0)   
 # 0:取列的最小值，1：取行的最小值  
 maxVals=dataSet.max(0)  
 ranges=maxVals-minVals  
 normDataSet=zeros(shape(dataSet))   
 # 用shape读取dataset的维度  
 m=dataSet.shape[0]   
 # shape[0]为行数，样本个数  
 normDataSet=dataSet-tile(minVals,(m,1))   
 # 数据集中每个特征减去每个特征对应的最小值  
 normDataSet=normDataSet/tile(ranges,(m,1))   
 # 除以最大值与最小值的差  
 return normDataSet,ranges,minVals  
normDataSet,ranges,minVals=autoNorm(datingDataMat)

#### 4. 将文本文件解析成词条向量

# 可以先使用python的string.split()函数  
mySent='This book is the best book on python or M.L. I have ever laid eyes upon'  
print mySent.split() # 此时标点符号也被当作了词的一部分  
# 可以使用正则表达式来切分句子，其中分隔符是除单词、数字外的任意字符串  
import re  
regEx=re.compile('\\W\*')   
# compile创建一个pattern，其中分隔符是除单词、数字外的任意字符串  
listOfTokens=regEx.split(mySent)  
# 上面两句等价于 listOfTokens=re.split(r'\W\*',mySent)  
listOfTokens=[tok for tok in listOfTokens if len(tok) > 0]   
# 去除空字符  
listOfTokens=[tok.lower() for tok in listOfTokens if len(tok)>0]   
# 将所有字符串转换为小写或大写 .upper()  
print listOfTokens

#### 5.测试算法：计算错误率

def datingClassTest():

hoRatio=0.10

# 取总样本的十分之一作为测试样本

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

normMat,ranges,minVals=autoNorm(datingDataMat)

m=normMat.shape[0]

numTestVecs=int(m\*hoRatio)

errorCount=0.0

for i in random.randint(0,m,numTestVecs):

# 在m个样本随机挑选出numTestVecs个作为测试样本,区间左开右闭（即索引为0到m-1）

# 得到的测试样本为 normMat[1:(numTestVecs-1),:]

# 训练样本就为 normMat[numTestVecs:m,:]

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))

for i in range(10): # 随机取10个样本作为测试样本  
 randIndex=int(random.uniform(0,len(trainingSet)))  
 # uniform:在区间(左闭右开)中取一随机数  
 # trainingSet是一个整数列表，其中的值从0到49，len(trainingSet)=50  
 testSet.append(trainingSet[randIndex])

random.shuffle(indexList)   
# 对indexList进行混洗（shuffle），从而实现数据集数据点的随机选取

#### 6.图像的处理

##### 1.图像的读取

传统方法;

from skimage import io  
import matplotlib  
import matplotlib.pyplot as plt  
img=io.imread('demo.jpg',as\_grey=True)  
io.imshow(img)  
plt.show()

opencv: conda install *--channel https://conda.anaconda.org/menpo opencv*

*此时安装的版本为2.4.11*

import matplotlib.pyplot as plt  
import numpy as np  
import cv2  
img = cv2.imread("demo.jpg")  
img = cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)  
plt.imshow(img)  
plt.axis("off")  
plt.show()

# # 做出原始图像与灰度图像  
# gray\_img = cv2.imread("demo.jpg",0)  
# orign\_img = cv2.imread("demo.jpg",1)  
# plt.subplot(121)  
# orign\_img = cv2.cvtColor(orign\_img,cv2.COLOR\_BGR2RGB)  
# plt.imshow(orign\_img)  
# plt.axis("off")  
# plt.subplot(122)  
# plt.imshow(gray\_img,cmap=plt.cm.gray)  
# plt.axis("off")  
# plt.show()

##### 将一个32\*32的二进制图像矩阵转换为1\*1024的向量

常规for循环：

def img2vector(filename):  
 returnVect=zeros((1,1024))  
 fr = open(filename)  
 for i in range(32):  
 lineStr=fr.readline()  
 # 循环读出文件的前32行,readline每次只读一行  
 for j in range(32):  
 returnVect[0,32\*i+j] = int(lineStr[j])  
 return returnVect

### 4.损失函数（代价函数）

一、0-1损失函数

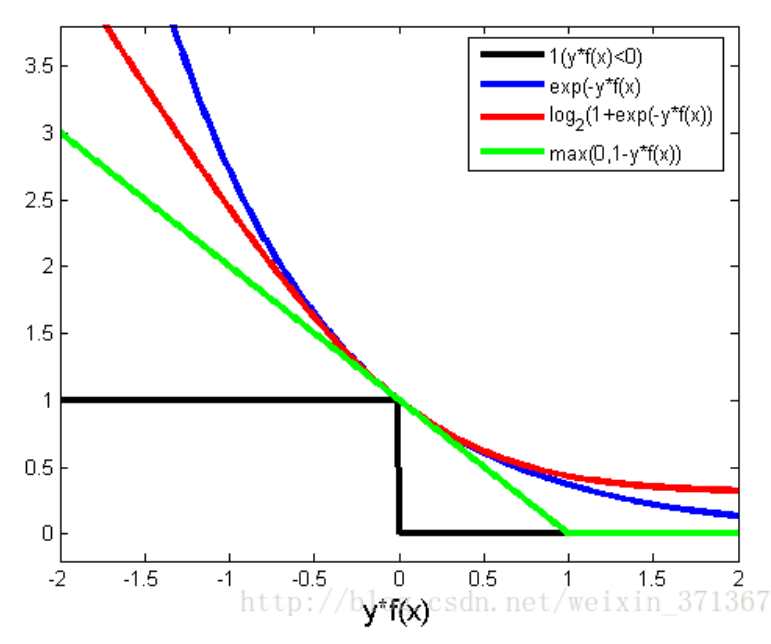
二、平方损失函数**（Square Loss）**：主要是最小二乘法（OLS）中；

三、绝对值损失函数

四、对数损失函数**（Cross Entropy Loss，Softmax Loss ）**：用于Logistic 回归与Softmax 分类中；

五、指数损失函数**（Exponential Loss）** ：主要用于Adaboost 集成学习算法中；

六、铰链损失函数（**Hinge Loss）**：主要用于支持向量机（SVM） 中；



一、0-1损失函数

  可以看出，该损失函数的意义就是，当预测错误时，损失函数值为1，预测正确时，损失函数值为0。该损失函数不考虑预测值和真实值的误差程度，也就是只要预测错误，预测错误差一点和差很多是一样的。

二、平方损失函数

  该损失函数的意义也很简单，就是取预测差距的平方。

三、绝对值损失函数

  该损失函数的意义和上面差不多，只不过是取了绝对值，差距不会被平方缩放。

四、对数损失函数

  这个损失函数就比较难理解了。事实上，该损失函数用到了极大似然估计的思想。P(Y|X)通俗的解释就是：在当前模型的基础上，对于样本X，其预测值为Y，也就是预测正确的概率。由于概率之间的同时满足需要使用乘法，为了将其转化为加法，我们将其取对数。最后由于是损失函数，所以预测正确的概率越高，其损失值应该是越小，因此再加个负号取个反。

经典的对数损失函数包括entropy和softmax，这里以entropy为例

在逻辑回归中，我们采用的是对数损失函数。由于逻辑回归是服从伯努利分布(0-1分布)的，并且逻辑回归返回的sigmoid值是处于(0,1)区间，不会取到0,1两个端点。因此我们能够将其损失函数写成以下形式：

  解释一下含义：

* 当真实值y=1时，，当预测值越接近1，也越接近最大值1，加上负号后就代表误差值最小。而当预测值越接近0，越接近负无穷，加上负号后就代表误差值最大。
* 当真实值y=0时，，当预测值越接近0，也越接近最大值1，加上负号后就代表误差值最小。而当预测值越接近1，越接近负无穷，加上负号后就代表误差值最大。

  最后逻辑回归中对于所有样本的损失函数为：

五、指数损失函数

指数误差，在boosting算法中比较常见:

学过Adaboost算法的人都知道，它是前向分步加法算法的特例，是一个加和模型，损失函数就是指数函数。在Adaboost中，经过m此迭代之后，可以得到

[$$f_m (x) = f_{m-1}(x) + \alpha_m G_m(x)$$](http://latex.codecogs.com/gif.latex?$$f_m%20(x)%20%3D%20f_%7bm-1%7d(x)%20+%20\alpha_m%20G_m(x)$$)

Adaboost每次迭代时的目的是为了找到最小化下列式子时的参数

[$$\arg \min_{\alpha, G} = \sum_{i=1}^{N} exp[-y_{i} (f_{m-1}(x_i) + \alpha G(x_{i}))]$$](http://latex.codecogs.com/gif.latex?$$\arg%20\min_%7b\alpha,%20G%7d%20%3D%20\sum_%7bi%3D1%7d%5e%7bN%7d%20exp%5b-y_%7bi%7d%20(f_%7bm-1%7d(x_i)%20+%20\alpha%20G(x_%7bi%7d))%5d$$)

而指数损失函数(exp-loss）的标准形式如下

[$$L(y, f(x)) = \exp[-yf(x)]$$](http://latex.codecogs.com/gif.latex?L(y,%20f(x))%20%3D%20\exp%5b-yf(x)%5d)

可以看出，Adaboost的目标式子就是指数损失，在给定n个样本的情况下，Adaboost的损失函数为：

[L(y, f(x)) = \frac{1}{n}\sum_{i=1}^{n}\exp[-y_if(x_i)]](http://latex.codecogs.com/gif.latex?L(y,%20f(x))%20%3D%20\frac%7b1%7d%7bn%7d\sum_%7bi%3D1%7d%5e%7bn%7d\exp%5b-y_if(x_i)%5d)

六、hinge loss损失函数

hinge损失函数和SVM是息息相关的。在线性支持向量机中，最优化问题可以等价于下列式子：  
[$$\min_{w,b}  \ \sum_{i}^{N} [1 - y_i(w\cdot x_i + b)]_{+} + \lambda||w||^2 $$](http://latex.codecogs.com/gif.latex?$$\min_%7bw,b%7d%20\%20\sum_%7bi%7d%5e%7bN%7d%20%5b1%20-%20y_i(w\cdot%20x_i%20+%20b)%5d_%7b+%7d%20+%20\lambda||w||%5e2%20$$)  
下面来对式子做个变形，令：  
[$$[1 - y_i(w\cdot x_i + b)]_{+} = \xi_{i}$$](http://latex.codecogs.com/gif.latex?$$%5b1%20-%20y_i(w\cdot%20x_i%20+%20b)%5d_%7b+%7d%20%3D%20\xi_%7bi%7d$$)  
于是，原式就变成了：  
[$$\min_{w,b}  \ \sum_{i}^{N} \xi_i + \lambda||w||^2 $$](http://latex.codecogs.com/gif.latex?$$\min_%7bw,b%7d%20\%20\sum_%7bi%7d%5e%7bN%7d%20\xi_i%20+%20\lambda||w||%5e2%20$$)

## Sklearn 机器学习算法包

## 其他相关内容

### 1.RSS

# RSS是站点用来和其他站点之间共享内容的一种简易方式（也叫聚合内容），通常被用于新闻和其他按顺序排列的网站，例如Blog

# universal feed parser是 Python 中最常用的RSS程序库，在Anaconda Prompt下输入：conda install feedparser安装，conda list检查库是否已安装

### 2.API

API 是一种为客户提供服务的方式。当你在浏览器中输www.facebook.com，一则请求会出现在Facebook的远程服务器上。一旦你的浏览器收到了响应，它就会解析代码、呈现出网页。

对于浏览器来说（这也是所谓的客户端），Facebook的服务器就是一个应用程序编程接口。这意味着每当你在互联网上访问一个页面的时候，你都在与某个远程服务器的API发生交互。

API并不完全等同于远程服务器——它其实是服务器的一部分，负责接收请求并发送响应。

http://www.sohu.com/a/243658082\_669829