## **交叉验证简介**

交叉验证的基本思想是把在某种意义下将原始数据(dataset)进行分组,一部分做为训练集(train set),另一部分做为验证集(validation set or test set),首先用训练集对分类器进行训练,再利用验证集来测试训练得到的模型(model),以此来做为评价分类器的性能指标。

### **什么时候用交叉验证**

假设未知模型具有一个或多个待定的参数，且有一个数据集能够反映该模型的特征属性（训练集）。

适应的过程是对模型的参数进行调整，以使模型尽可能反映训练集的特征。

如果从同一个训练样本中选择独立的样本作为验证集合，当模型因训练集过小或参数不合适而产生过拟合时，验证集的测试会予以反映。

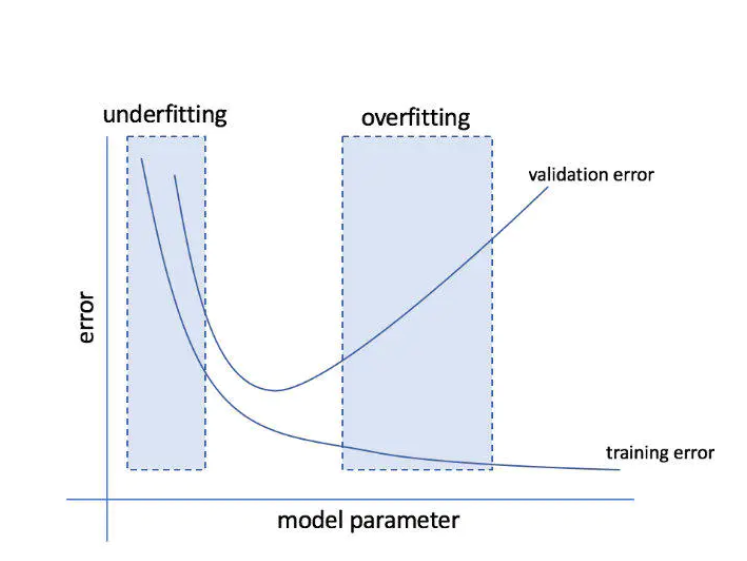
总结：交叉验证是一种预测模型拟合性能的方法

**交叉验证的目标**

确定一个原数据集的子集，去限制ML模型在训练阶段的一些问题，比如模型的过拟合、欠拟合等，同时提供了一种判断标准去衡量模型在独立数据集上的泛化能力。

欠拟合（Underfitting）  
是指模型不能获取数据集的主要信息，在训练集及测试集上的表示都十分糟糕。

过拟合（Overfitting）  
是指模型不仅获取了数据集的信息还提取了噪声数据的信息是的模型在训练集有非常好的表现但在测试集上的表现及其糟糕



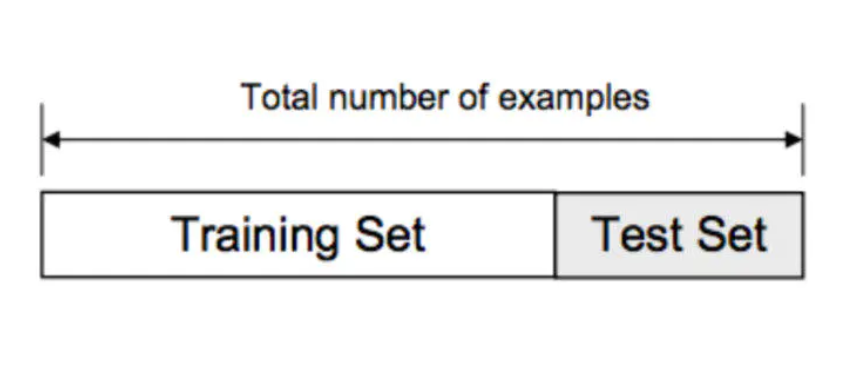
**多种交叉验证方法**

Train/Test split（Groups=2）

在这种方法中，将数据集划分为两个部分：训练集和测试集。

在训练集和测试集之前没有交叉重叠的样本，如果训练集和测试集之间出现了交叉重叠的样本，以此训练得到模型是不可靠的。

将原始数据随机分为两组，一组做为训练集，一组做为验证集，利用训练集训练分类器，然后利用验证集验证模型，记录最后的分类准确率为此分类器的性能指标。



好处：

处理简单，只需随机把原始数据分为两组即可

坏处：

没有达到交叉的思想，随机的将原始数据分组，所以最后验证集分类准确率的高低与原始数据的分组有很大的关系，得到的结果并不具有说服性。  
容易导致过拟合，数据集的划分不是随机进行的，或者划分出的子集只包含了数据集的一部分特征。

在Python中使用：xtrain,xtest,ytrain,ytest=sklearn.model\_selection.train\_test\_split()实现  
  
data要进行划分的数据集，支持列表、数据帧、数组、矩阵.  
test\_size 测试集所占比例  
train\_size训练集所占比例  
random\_state随机数种子，用于生成重复随机数  
shuffle 是否在划分数据集之前打乱数据集

2-折交叉验证

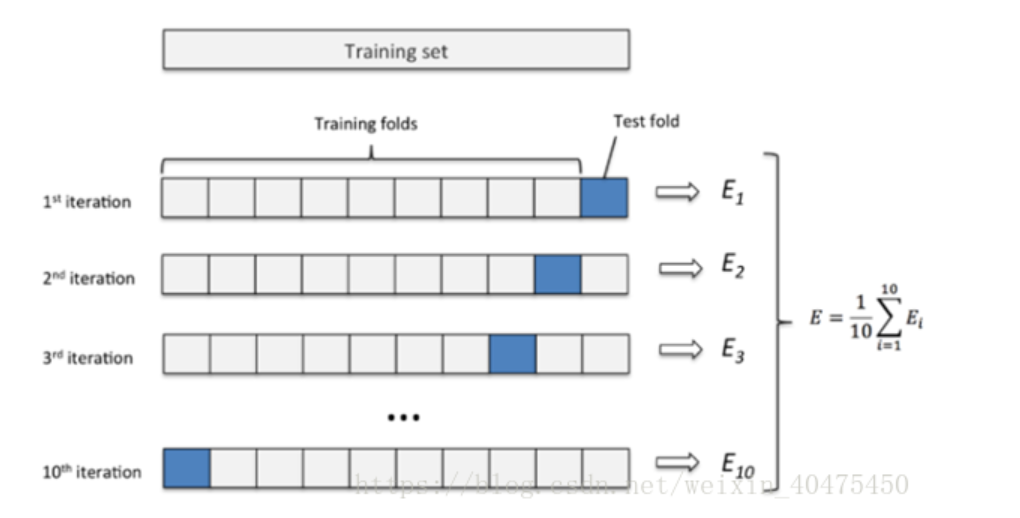
将数据集分成两个相等大小的子集，进行两次的分类器训练。在第一次中，一个子集作为训练集，另一个作为测试集；在第二次训练中，将训练集与测试集对换。

训练集样本数太少，不足以代表原本样本的分布，导致测试阶段辨识率容易出现明显落差。

K-折交叉验证

将原始数据分成K组，将每个子集数据分别做一次验证集，其余的K-1组子集数据作为训练集，这样会得到K个模型，用这K个模型最终的验证集的分类准确率的平均数作为此分类器的性能指标。

K一般大于等于2。



可以有效的避免过拟合与欠拟合的发生，最后得到的结果也比较具有说服性。

当我们需要对一些小的数据集进行统计分析时，K-Flod是一个好的选择，在进行k次交叉验证时能获得足够多的模型的质量差异和不同的最佳参数。一般来说，经过长期的经验积累，我们选择k=5或k=10IMG_257，以产生既不受过高偏差也不受过高偏差影响的测试误差估计。

在Python中使用：sklearn.model\_selection.KFlod()

n\_split折叠数量，默认为3。  
shuffle是否在分割成批次之前打乱数据集，默认不打乱。  
random\_state随机数种子，在进行打乱数据操作时使用。

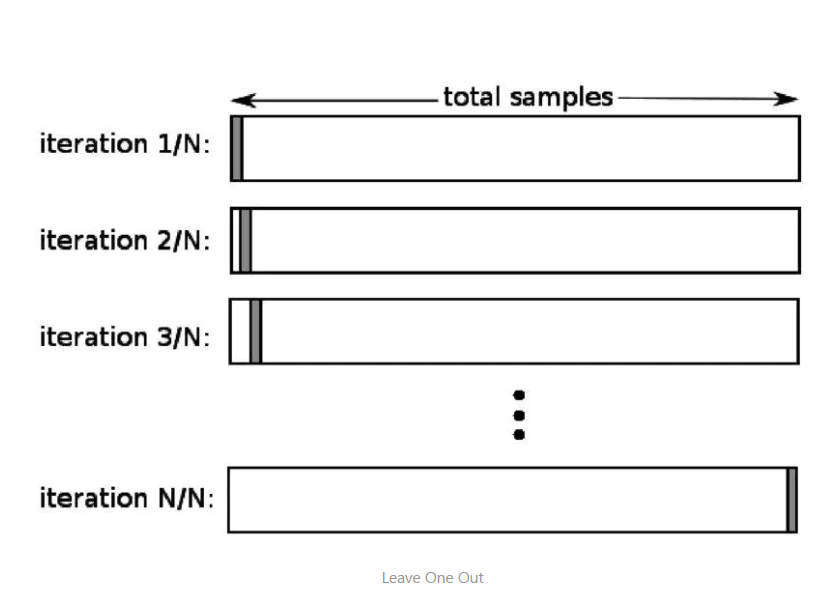
十折交叉验证

将数据集分成十份，轮流将其中9份作为训练数据，1份作为测试数据，进行试验。每次试验都会得出相应的正确率。

10次的结果的正确率的平均值作为对算法精度的估计，一般还需要进行多次10折交叉验证（例如10次10折交叉验证），再求其均值，作为对算法准确性的估计

留一交叉验证

在数据缺乏的情况下使用，如果设原始数据有N个样本，每个样本单独作为验证集，其余的N-1个样本作为训练集，会得到N个模型，用这N个模型最终的验证集的分类准确率的平均数作为此分类器的性能指标。



优点：

每一回合中几乎所有的样本皆用于训练模型，因此最接近原始样本的分布，这样评估所得的结果比较可靠。

实验过程中没有随机因素会影响实验数据。

缺点：

计算成本高，需要建立的模型数量与原始数据样本数量相同。

在Python中使用：sklearn.model\_selection.LeaveOneOut()=KFlod(n\_split=n)  
Methods:

get\_n\_splits(X)返回交叉验证器中的拆分迭代次数，

,x为训练数据，形状为(n\_samples,n\_features)  
split(x,y,groups) 生成索引已将数据集拆分为训练集和测试集，

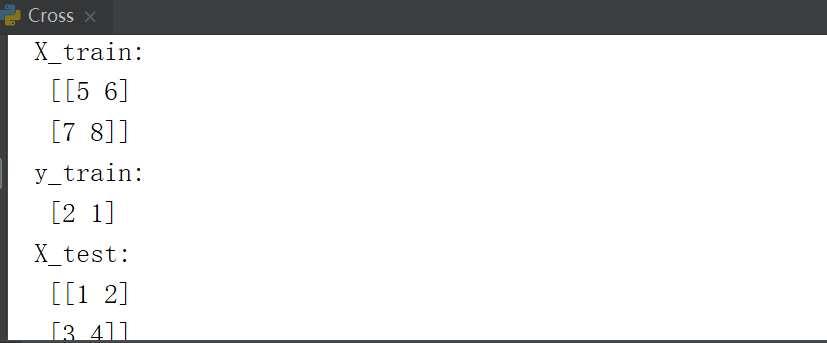
X为训练集，y为标签，groups为拆分时使用的组标签

**Cross.py**

### **Holdout 验证**

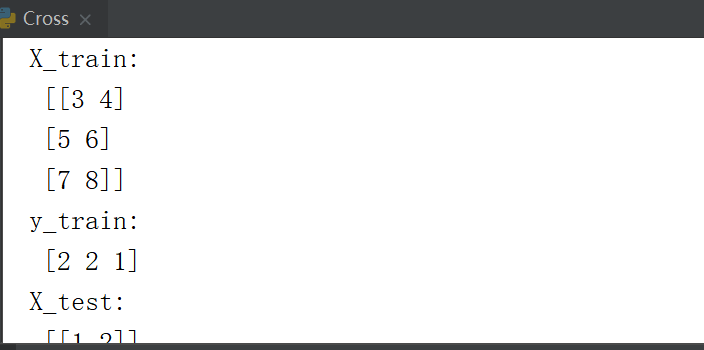
**test\_size=0.50**

将原始数据随机分为两组，一组做为训练集，一组做为验证集，利用训练集训练分类器，然后利用验证集验证模型.





**test\_size=0.10**

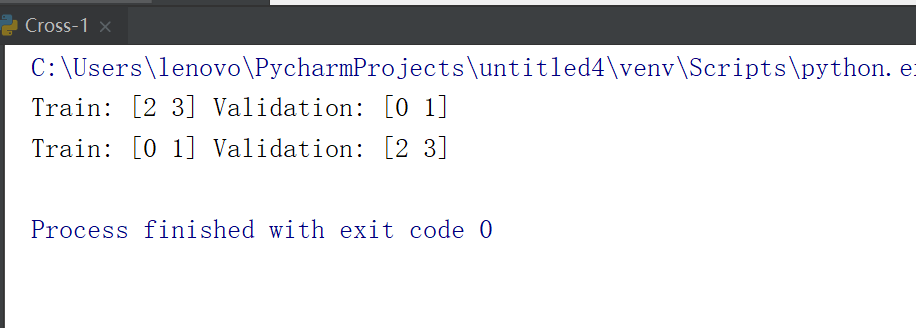


**Cross-1.py**

1. **fold cross-validation**

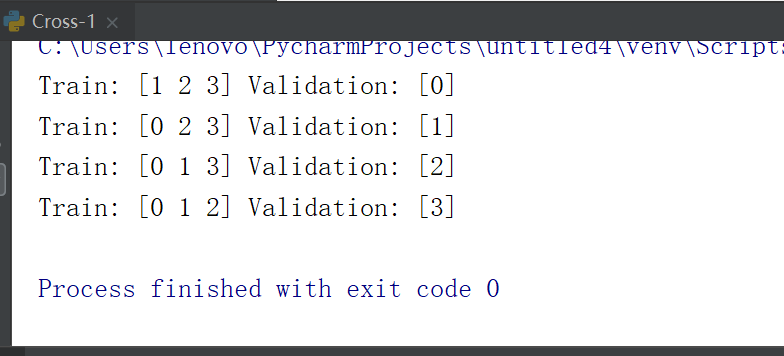
K次交叉验证，将训练集分割成K个子样本，一个单独的子样本被保留作为验证模型的数据，其他K-1个样本用来训练。交叉验证重复K次，每个子样本验证一次，平均K次的结果或者使用其它结合方式，最终得到一个单一估测。

代码输出如下：



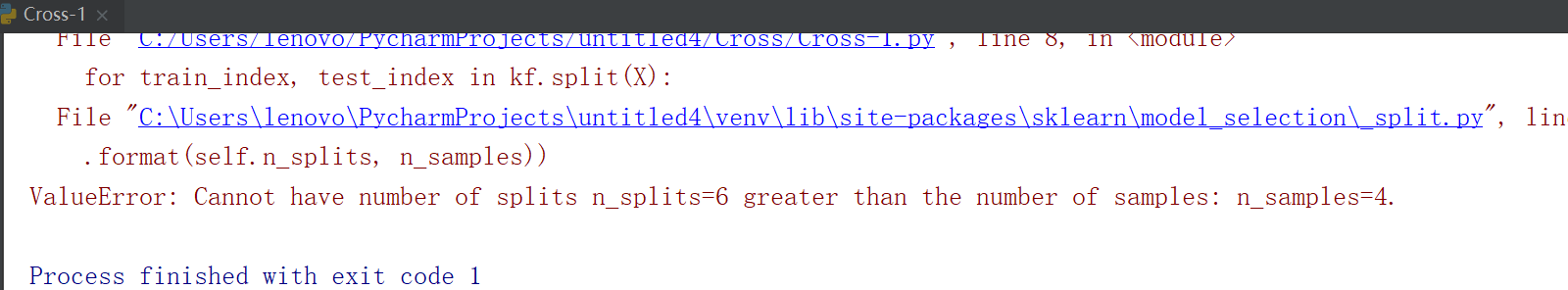
改变其中的数据值，kf = KFold(n\_splits=4)

输出改变如下：



接下来，又把数字增多，kf = KFold(n\_splits=6)

代码报错如下：



分割的数量不能大于样本的数量

**Cross-2.py**

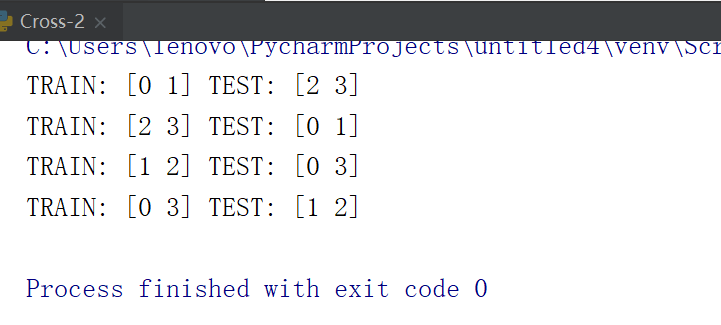
### **RepeatedKFold  p次k折交叉验证**

在实际当中，只进行一次k折交叉验证是不够的，需要进行多次，RepeatedKFold方法可以控制交叉验证的次数，在每次重复中产生不同的分裂。

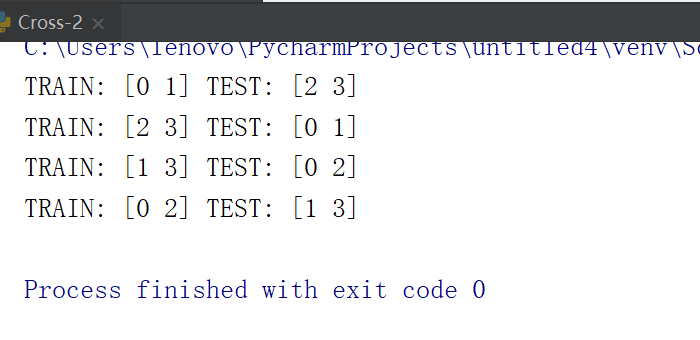
n\_repeats重复几次  
random\_state随机种子  
random\_state 相当于随机数种子random.seed() random\_state 与 random seed 作用是相同的

如果设置了相同的 random seed（数值），它们取的随机数就完全相同，多运行几次也是这样

代码输出如下所示：



更换了random\_state的数值以后，输出值发生了改变：



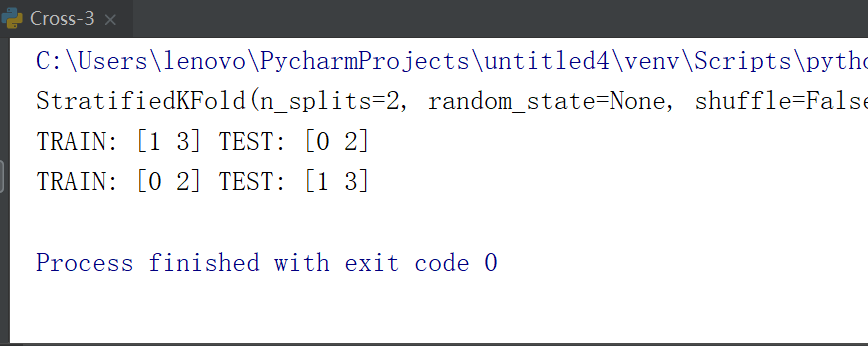
**Cross-3.py**

**Stratified k-fold**

与k-fold类似，将数据集划分成k份，划分的k份中，每一份内各个类别数据的比例和原始数据集中各个类别的比例相同。

StratifiedKFold是针对非平衡数据的分层采样。分层采样就是在每一份子集中都保持原始数据集的类别比例。例如原始数据集正类：负类=3:1，这个比例也要保持在各个子集中才行。

shuffle是否在分割成批次之前打乱数据集，默认不打乱。



**Cross-4.py**

**Leave-One-Out Cross Validation**

留一验证只使用原本样本中的一项来当做验证资料， 而剩余的则留下来当做训练资料。 这个步骤一直持续到每个样本都被当做一次验证资料。

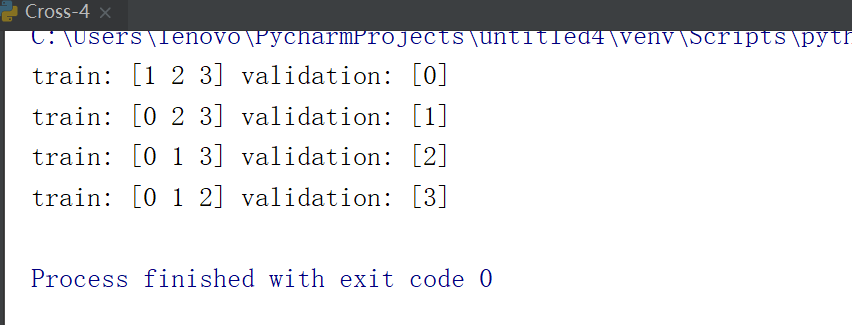
相比于KFold、LOO有两个明显的优点：

每一回合中几乎所有的样本皆用于训练模型，最接近原始样本的分布，评估所得的结果比较可靠

实验过程中没有随机因素会影响实验数据。

缺点是：计算成本高，因为需要建立的模型数量与原始数据样本数据相同。

代码输出如下所示：



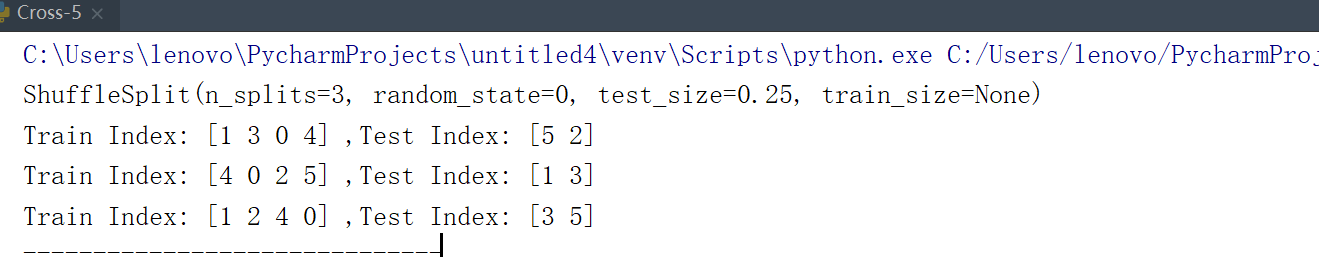
**Cross-5.py**

**ShuffleSplit**

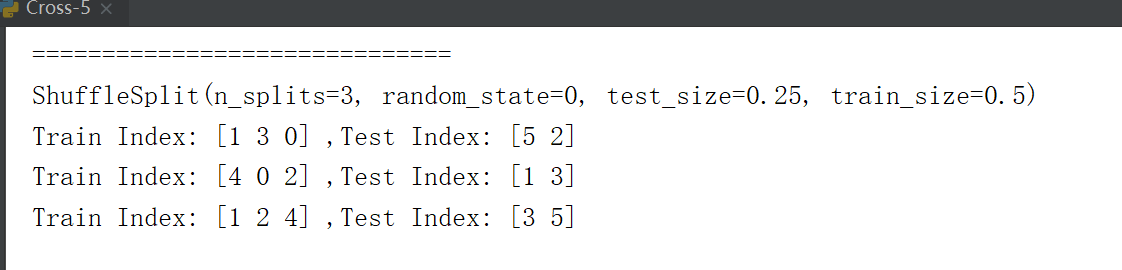
ShuffleSplit 把数据集打乱顺序，然后划分测试集和训练集，训练集额和测试集的比例随机选定，训练集和测试集的比例的和可以小于1。

可以通过random\_state这个种子来重现我们的分配方式，如果没有指定，那么每次都是随机的。

代码输出如下所示：



设置了train\_size的值，代码输出如下：



**其它特殊情况的数据划分方法**

对于分类数据来说，它们的数据是可能分配是不均匀的，比如在医疗数据当中得癌症的人比不得癌症的人少很多，这个时候，使用的数据划分方法有 StratifiedKFold  ，Stratified，ShuffleSplit

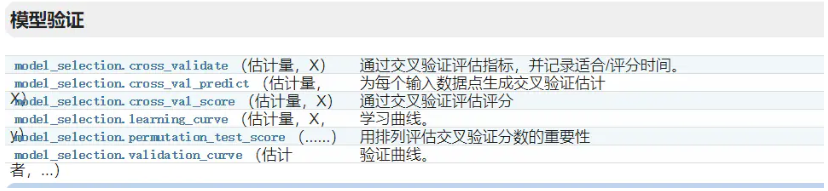
对于分组数据来说，它的划分方法是不一样的，主要的方法有 GroupKFold，LeaveOneGroupOut，LeavePGroupOut，GroupShuffleSplit

对于时间关联的数据，方法有TimeSeriesSplit

## **运用交叉验证进行模型评估**

上面使用交叉验证进行数据集的划分。我们用交叉验证的方法并且结合一些性能度量方法来评估模型好坏，我们可以直接使用sklearn当中提供的交叉验证评估方法，这些方法如下：

使用的数据，是sklearn中自带的数据集



**Cross-6.py**

### **cross\_value\_score**

使用交叉验证来计算模型的评分情况

clf是使用的算法

cv=5表示cross\_val\_score采用的是k-fold cross validation的方法，重复5次交叉验证

cv是使用的交叉验证的生成器或者迭代器，它决定了交叉验证的数据是如何划分的，当cv的取值为整数的时候，使用(Stratified)KFold方法

**sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache\_size=200, class\_weight=None, verbose=False, max\_iter=-1, decision\_function\_shape=None,random\_state=None)**

参数：

C：C-SVC的惩罚参数，C的默认值是1.0

C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。

kernel ：核函数，默认是rbf，可以是‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’

degree ：多项式poly函数的维度，默认是3，选择其他核函数时可以忽略。

gamma ： ‘rbf’,‘poly’ 和‘sigmoid’的核函数参数。默认是’auto’，有时候可以选择1/n\_features

coef0 ：核函数的常数项。对于‘poly’和 ‘sigmoid’有用。

probability ：是否采用概率估计.默认为False

决定是否启用概率估计。需要在训练fit()模型时加上这个参数，之后才能用相关的方法：predict\_proba和predict\_log\_proba

shrinking ：是否采用shrinking heuristic方法，默认为true

tol ：停止训练的误差值大小，默认为1e-3

cache\_size ：核函数cache缓存大小，默认为200

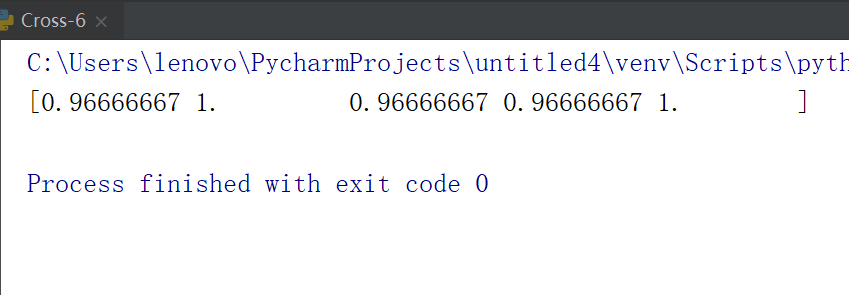
class\_weight ：类别的权重，字典形式传递。

verbose ：是否允许冗余输出

max\_iter ：最大迭代次数。-1为无限制。

decision\_function\_shape ：‘ovo’, ‘ovr’ or None, default=None3

random\_state ：数据洗牌时的种子值，为int值

代码输出如下所示：（准确率）  


**Cross-7.py Cross-8.py**

### **cross\_validate**

cross\_validate方法和cross\_validate\_score有个两个不同点：它允许传入多个评估方法，可以使用两种方法来传入，一种是列表的方法，另外一种是字典的方法。

最后返回的scores为一个字典，字典的key为：dict\_keys(['fit\_time', 'score\_time', 'test\_score', 'train\_score'])

允许指定多个指标进行评估

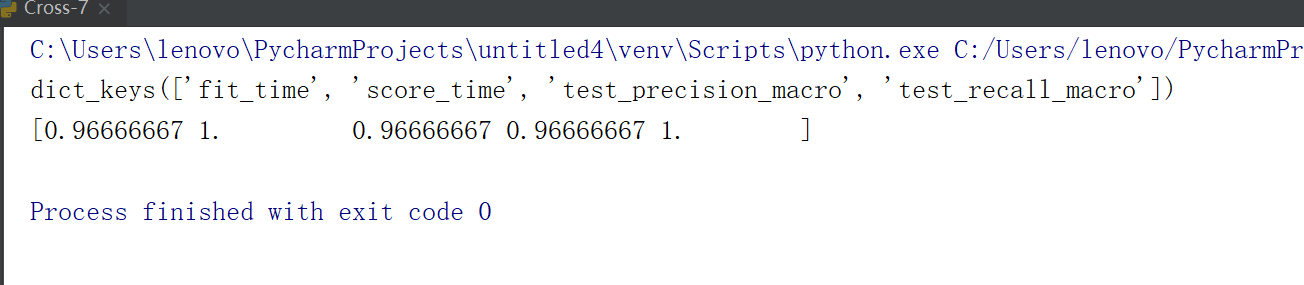
除了测试得分之外，它还会返回一个包含训练得分，拟合次数， score-times （得分次数）的一个字典。

对于单个度量评估，其中 scoring 参数是一个字符串，可以调用或 None ， keys 将是 - [‘test\_score’, ‘fit\_time’, ‘score\_time’]

而对于多度量评估，返回值是一个带有以下的 keys 的字典 - [‘test\_<scorer1\_name>’, ‘test\_<scorer2\_name>’, ‘test\_<scorer…>’, ‘fit\_time’, ‘score\_time’]

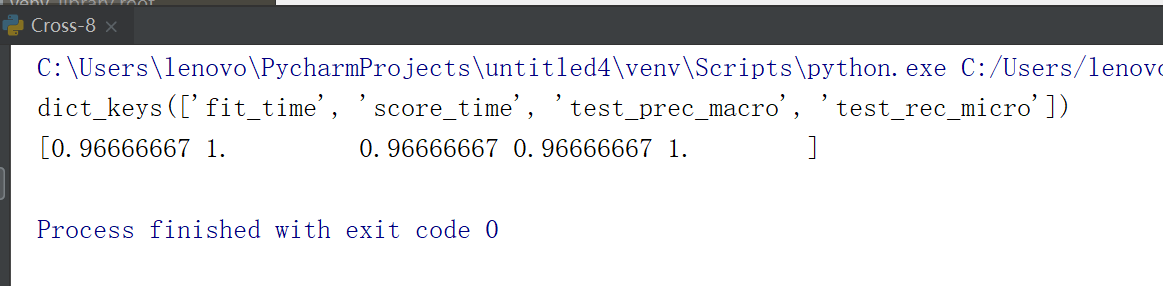
return\_train\_score 默认设置为 True 。 它增加了所有 scorers(得分器) 的训练得分 keys 。如果不需要训练 scores ，则应将其明确设置为 False 。

当scoring传入列表的时：



当scoring传入字典的时候如下：

作为一个字典mapping的分歧名称预定义或自定义得分函数



### 

**Cross-9.py**

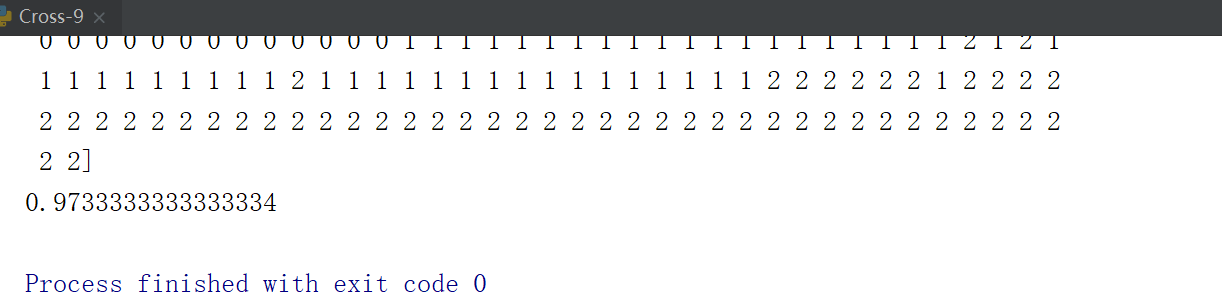
### **cross\_val\_predict**

cross\_val\_predict 和 cross\_val\_score的使用方法是一样的，但是它返回的是一个使用交叉验证以后的输出值，而不是评分标准。

运行过程：使用交叉验证的方法来计算出每次划分为测试集部分数据的值，直到所有的数据都有了预测值。

Eg:假如数据划分为[1,2,3,4,5]份，它先用[1,2,3,4]训练模型，计算出来第5份的目标值，然后用[1,2,3,5]计算出第4份的目标值，直到都结束为止。

代码输出如下所示：



### **自助法**

还有一种方法，那就是自助法

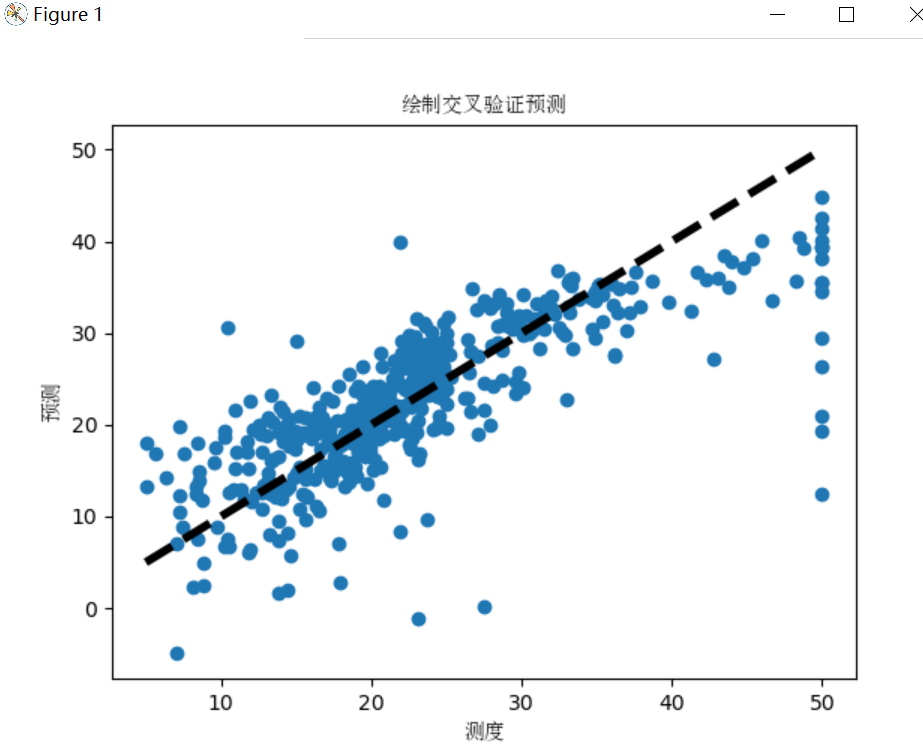
基本思想：对于含有m个样本的数据集D，我们对它进行有放回的采样m次，最终得到一个含有m个样本的数据集D',这个数据集D'会有重复的数据,我们把它用作训练数据。按照概率论的思想，在m个样本中，有1/e的样本从来没有采到，将这些样本即D\D'当做测试集。

**Cross-10.py**

### **用[scikit-learn绘制交叉验证预测](https://www.cnblogs.com/taceywong/p/5930498.html)实例**

### **使用cross\_val\_predict来可视化预测**

使用自带的boston数据



**Error**:关于matplotlib的，显示中文错误

安装python -m pip install matplotlib

使用matplotlib的字体管理器指定字体文件

plt.rcParams['font.sans-serif']=['SimHei']

#指定默认字体 SimHei为黑体  
plt.rcParams['axes.unicode\_minus']=False

#用来正常显示负号

用以下方法可列出Python可用的系统字体名称及路径：

from os import path

from matplotlib.font\_manager import fontManager

for i in fontManager.ttflist:

print(i.fname, i.name)

查出安装路径：

Import matplotlib

Print(matplotlib.matplotlib\_fname())

修改matplotlib默认字体的配置

找到mpl-data/matplotlibrc文件打开，删除font.family钱前面的注释

删除font.san-serif前面的注释，在这个配置的前面加上Microsoft YaHei

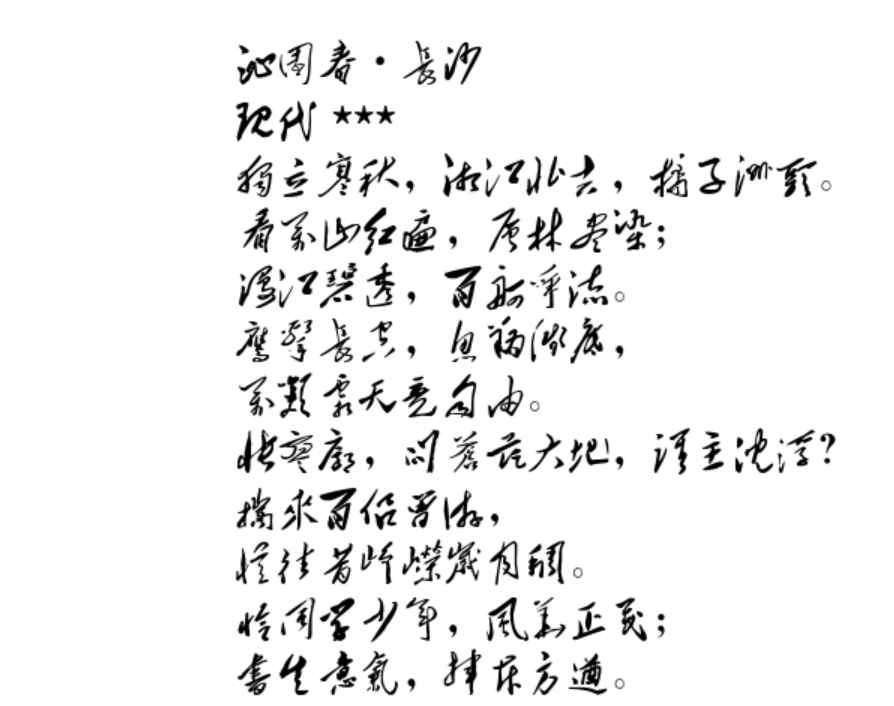
找到前面的kNN-1.py文件中，引入中文字体的方式和路径是：

font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)

找到安装的Matplotlib的路径  
font = FontProperties(fname=r"C:\Users\lenovo\PycharmProjects\untitled4\venv\Lib\site-packages\matplotlib\mpl-data\fonts\ttf\XXX.ttf", size=20)

**Cross-11.py**

**新下载了一个字体，试用是否正确**

****

**Cross-12.py**

**用名字测试了一下，哈哈（挺好看）**

****

**Cross-13.py**

解析：from scipy import interp

python中的插值，scipy-interp

f1=interp1d(x,y,kind='linear')#线性插值 f2=interp1d(x,y,kind='cubic')#三次样条插值

numpy.interp()主要使用场景为一维线性插值

一维线性插值.  
返回离散数据的一维分段线性插值结果.  
   
参数  
x: 数组

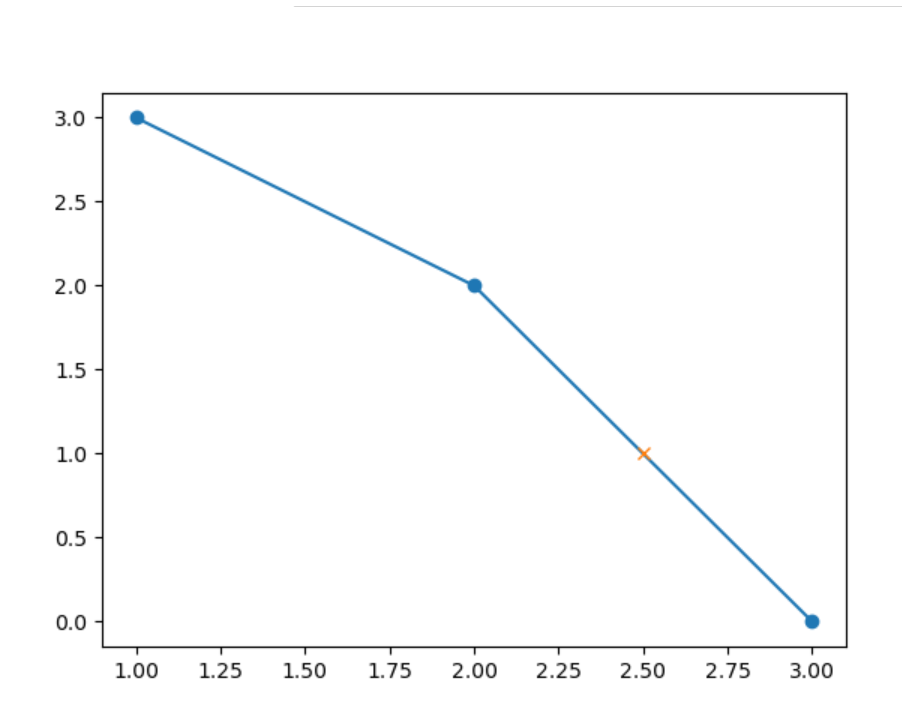
待插入数据的横坐标.  
   
xp: 一维浮点数序列  
原始数据点的横坐标，如果period参数没有指定那么就必须是递增的。否则，在使用xp = xp % period正则化之后，xp在内部进行排序.  
   
fp: 一维浮点数或复数序列  
原始数据点的纵坐标，和xp序列等长.  
   
left: 可选参数，类型为浮点数或复数（对应于fp值），当x < xp[0]时的插值返回值，默认为fp[0].  
   
right: 可选参数，类型为浮点数或复数（对应于fp值），当x > xp[-1]时的插值返回值，默认为fp[-1].

period: None或者浮点数，可选参数. 横坐标的周期. 此参数使得可以正确插入angular x-coordinates. 如果该参数被设定，那么忽略left参数和right参数。  
   
返回值  
浮点数或复数（对应于fp值）或ndarray. 插入数据的纵坐标，和x形状相同。

**做一个实例，图片化理解一下：**

**Cross-15.py**

结果输出如下：

****

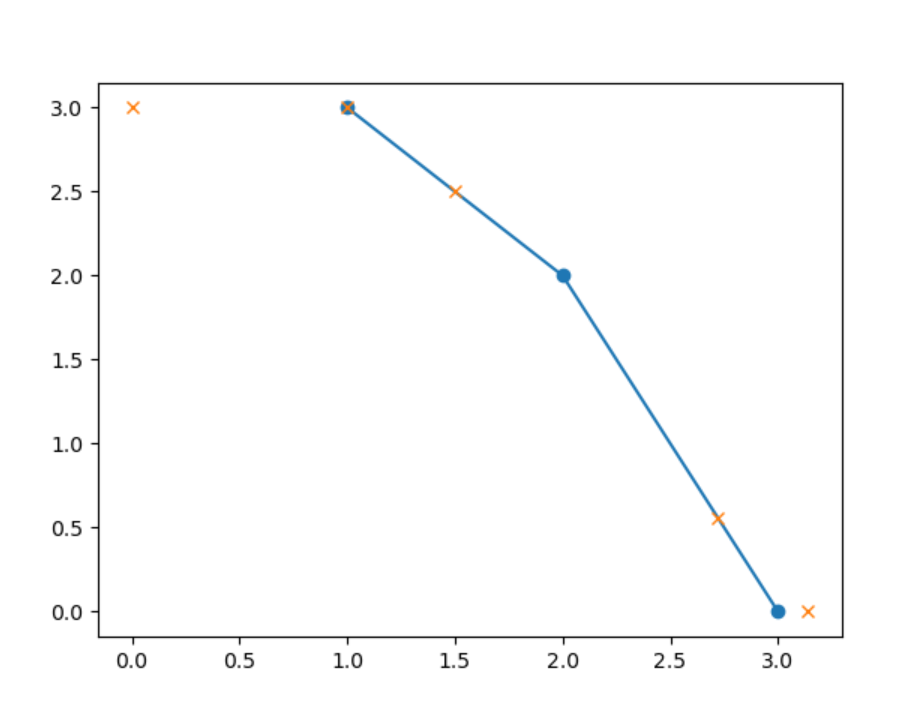
可以看到当x等于2.5时，在蓝色折线上可以找到y=1.0为插入坐标的纵坐标。

**Cross-16.py**

x的序列为[0, 1, 1.5, 2.72, 3.14]时，

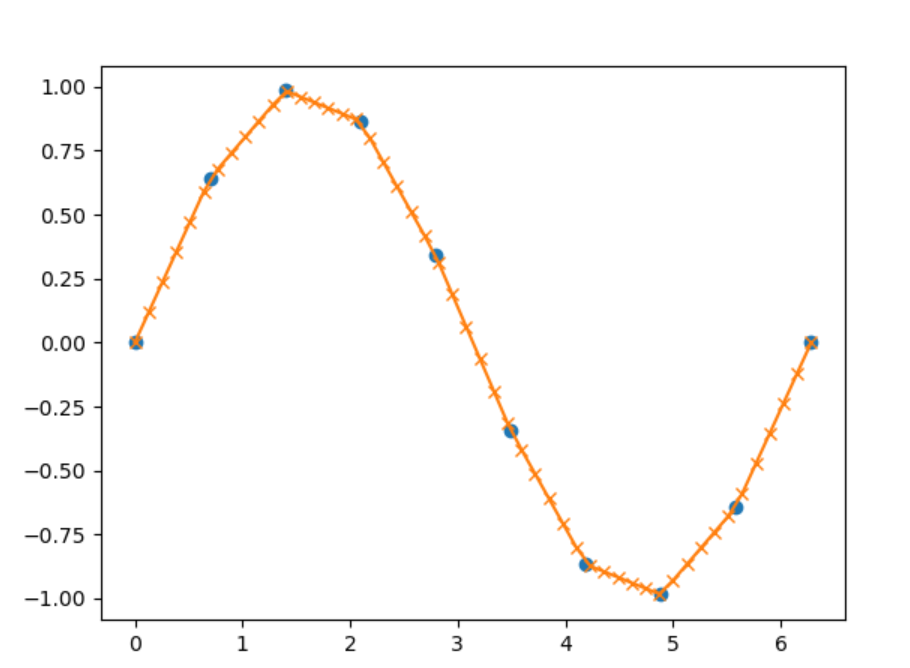
得到的y序列是[ 3. , 3. , 2.5 , 0.56, 0. ]。

需要注意的是当x=0时，由于0 < xp[0]，所以y被设定为fp[0]=3，当x=3.14时，由于3.14 > xp[-1]，则y被设定为fp[-1]=0。



**Cross-17.py**

在正弦函数图像上进行插值，黄色叉叉连成的是一段段折线，利用线性插值对正弦函数进行逼近的效果



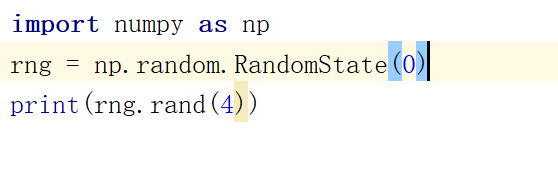
numpy.random.RandomState()是一个伪随机数生成器。

伪随机数是用确定性的算法计算出来的来自[0,1]均匀分布的随机数序列。并不真正的随机，但具有类似于随机数的统计特征，如均匀性、独立性等。

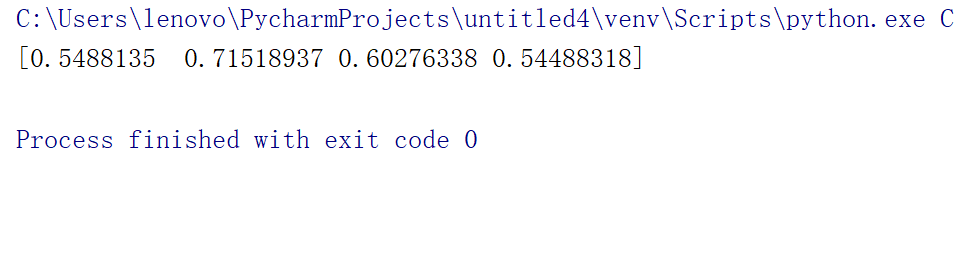
必须在rng这个变量下使用，如果不这样做，那么就得不到相同的随机数组了,

np.random.rand（）在默认状态下，是从默认随机数组里挑出的随机样本。

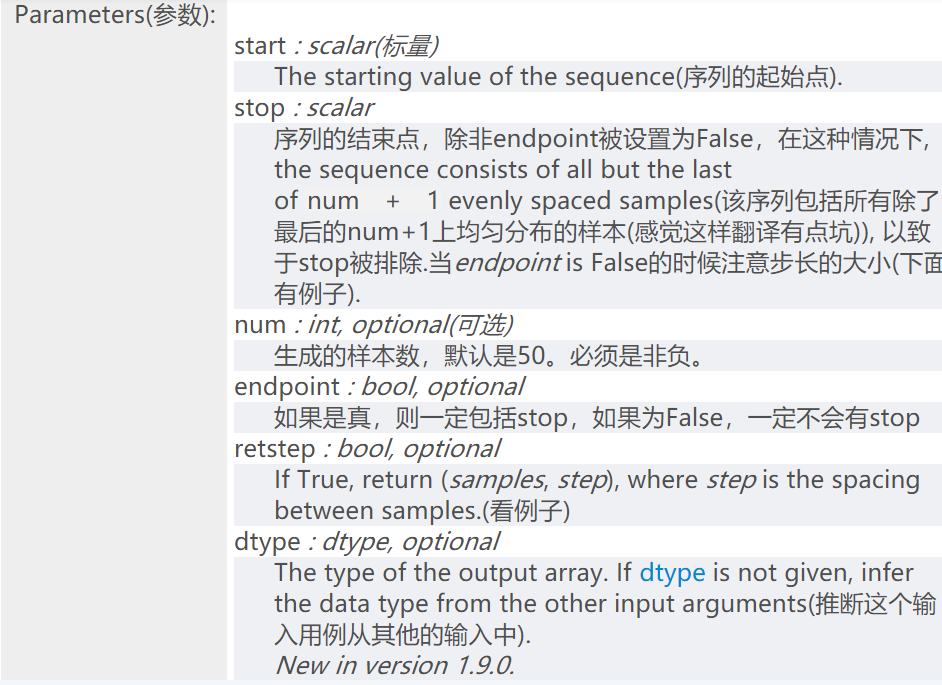
代码如下:

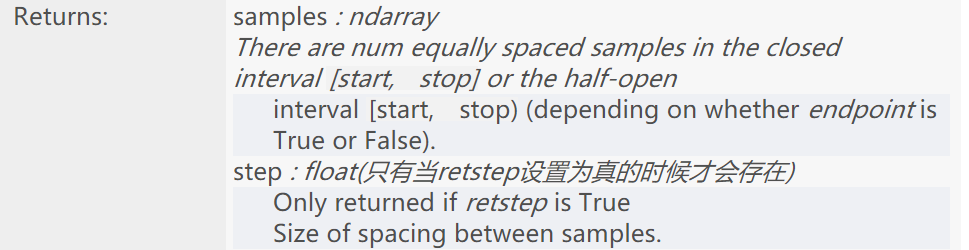


代码输出如下：



numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)





**Eg:**

import numpy as np  
np.linspace(1, 10, 10)  
array([  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.])

**Cross-18.py**

**错误率和精度**

错误率：在分类任务中，错分的样本数占样本总数的比例称为错误率。

精度 = 1 - 错误率。

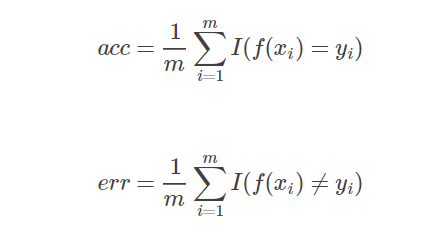
错误率和精度，是分类任务中最常用的两种性能度量，即适用于二分类任务，也适用于多分类任务。

精度的局限性:  
虽然模型的整体分类精度高，但不代表对占比小的类别的分类精度也高。

解决方案：

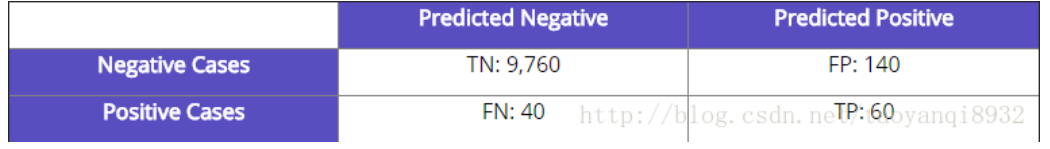
平均精度——每个类别下的样本精度的算术平均。

最常见的的分类中所用的度量是：accuracy(准确度)，error rate



准确度将每个类看得同等的重要，因此不适合用来分析不平衡的数据集，不平衡的数据集即正类样本远远小于负类的样本，，因此引入了下面的几种度量方法。

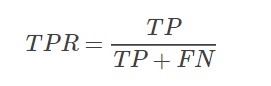
假定在10000个样本中有100个正样本，其余为负样本，分类器下的混淆矩阵(confusion matrix)为



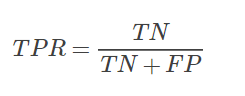
定义：

TN / True Negative: case was negative and predicted negative   
TP / True Positive: case was positive and predicted positive   
FN / False Negative: case was positive but predicted negative   
FP / False Positive: case was negative but predicted positive

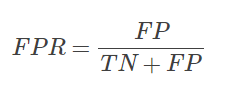
真正率(true positive rate,TPR)定义为被模型正确预测的正样本的比例：



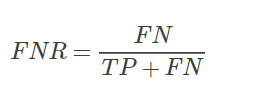
真负率(true negative rate,TFR)，定义为被模型正确预测的负样本的比例：



同理，假正率(false positive rate,FPR)



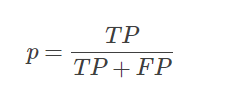
假负率(flase negative rate,FNR)



**Cross-19.py**

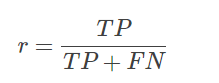
precision(精度)，也称为查准率:

其用于衡量模型避免错误的能力。

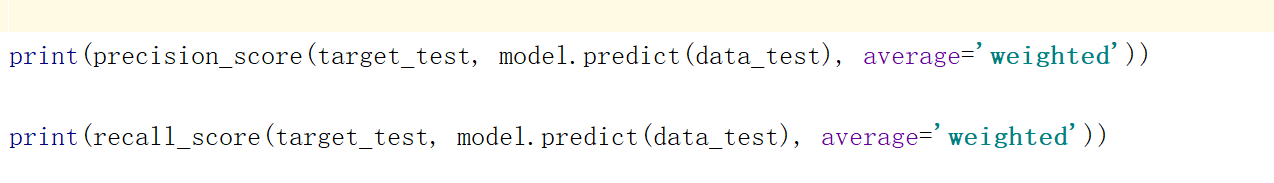


recall(召回率),也称为查全率：

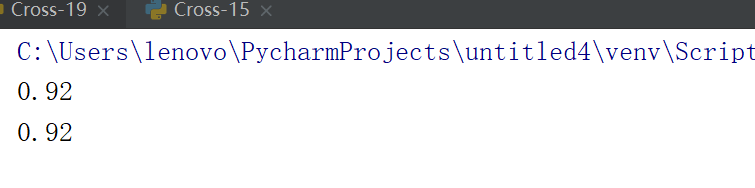
其用于衡量模型避免缺漏的能力。



计算召回率和精度：



代码输出如下：

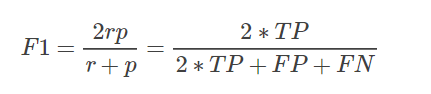


precision\_score 和 recall\_score 方法默认用来计算二分类问题，若要计算多分类问题，则需要设置 average 参数。

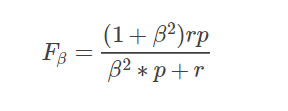
**Cross-20.py**

精度是确定分类器中断言为正样本的部分其实际中属于正样本的比例，精度越高则假的正例就越低，召回率则是被分类器正确预测的正样本的比例。

两者是一对矛盾的度量，其可以合并成令一个度量，F1度量：

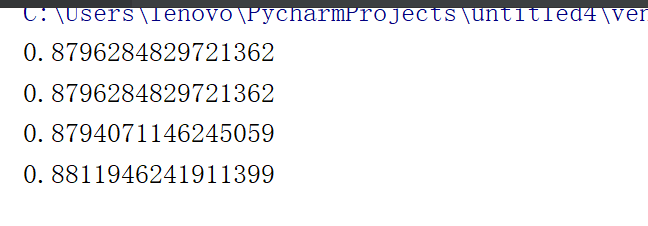


如果对于precision和recall的重视不同，则一般的形式：



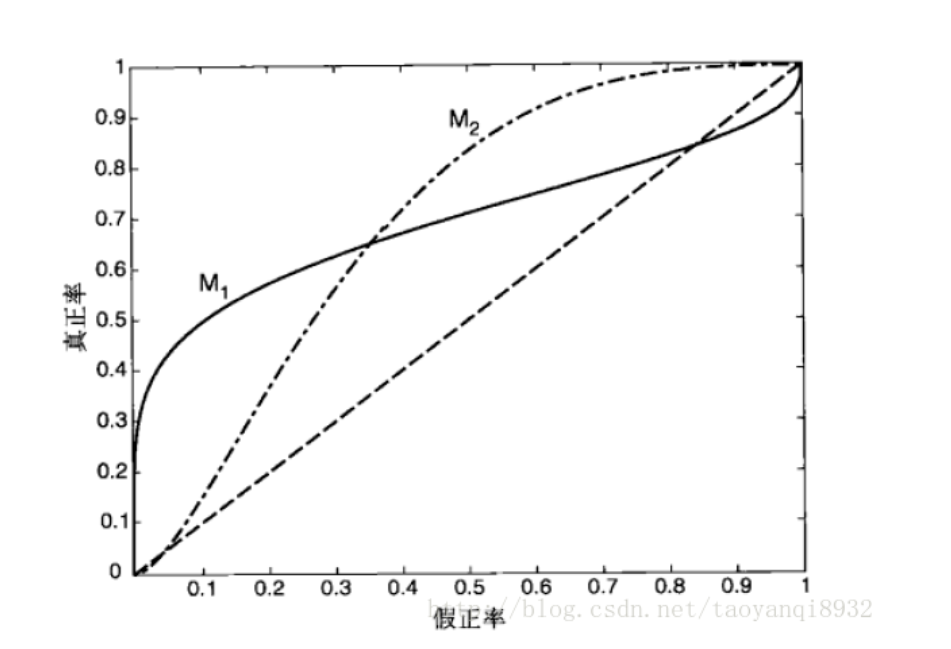
β=1则退化成F1,β>1则recall有更大影响，反之则precision更多影响。

计算F1和Fβ​



ROC--receiver operating characteristic

接受者操作特征，其显示的是分类器的真正率和假正率之间的关系



ROC曲线有助于比较不同分类器的相对性能.  
ROC曲线的面积为AUC(area under curve)，其面积越大则分类的性能越好，理想的分类器auc=1。

ROC曲线通常在Y轴上具有真阳性率，在X轴上具有假阳性率。这意味着图的左上角是“理想”点 - 误报率为零，真正的正率为1。这不太现实，但它确实意味着曲线下面积（AUC）通常更好。

ROC曲线通常用于二分类以研究分类器的输出。为了将ROC曲线和ROC区域扩展到多类或多标签分类，有必要对输出进行二值化。

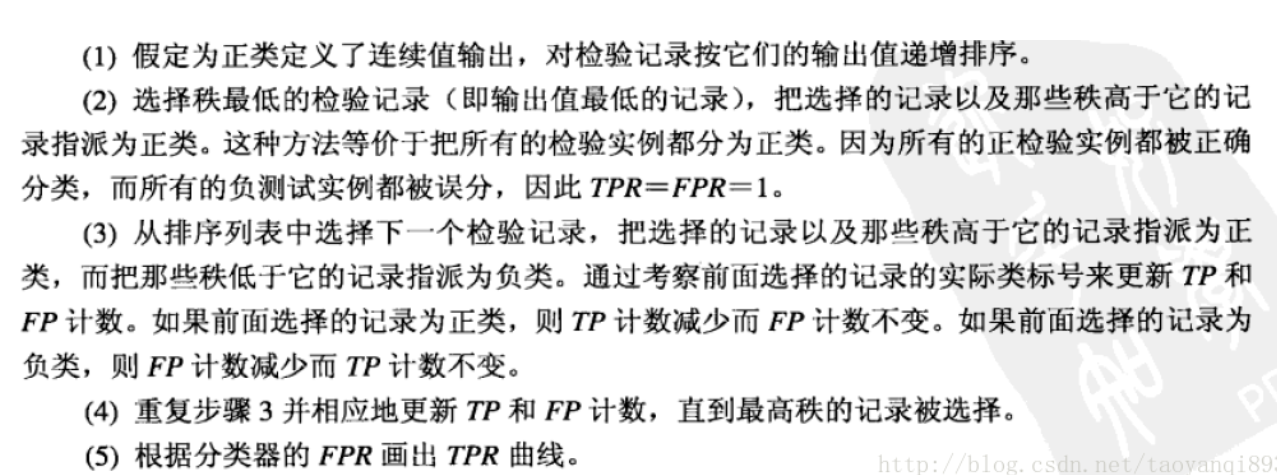
⑴可以每个标签绘制一条ROC曲线。

⑵可以通过将标签指示符矩阵的每个元素视为二元预测（微平均）来绘制ROC曲线。

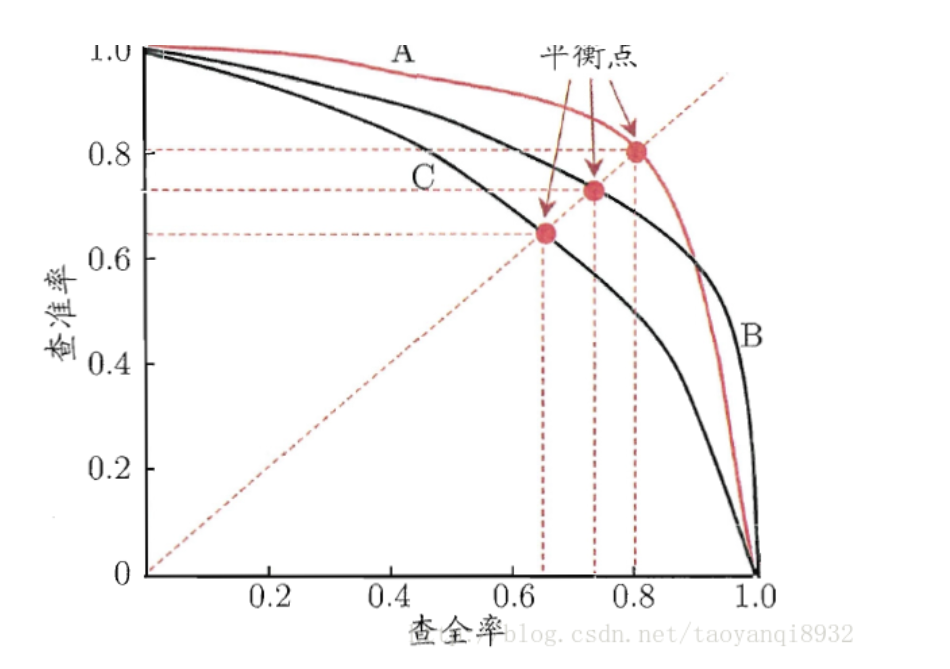
⑶另一种用于多类别分类的评估方法是宏观平均，它对每个标签的分类给予相同的权重。

**如何绘制ROC曲线**

分类器应该能输出连续的值，比如在逻辑回归分类器中，其以概率的形式输出，可以设定阈值大于0.5为正样本，否则为负样本。因此设置不同的阈值就可以得到不同的ROC曲线中的点。   
实现过程如下：



PR(precision recall)曲线表现的是precision和recall之间的关系，如图所示：



P-R 图可以用于学习器的比较：

若一个学习器的 P-R 曲线被另一个学习器的 P-R 曲线完全“包住”，则可断言后者的性能优于前者。

若两个学习器的 P-R 曲线发生了交叉，则难以一般性地断言两者孰优孰劣，只能在具体的查准率或查全率条件下进行比较。

**Cross-13.py**

**代码最终输出如下：**

