前面的已经学过了很多种不同的分类器，它们各有优缺点。

我们可以将不同的分类器组合起来，而这种组合结果则被成为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，也可以是同一种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。

# 

# **集成方法**

集成方法通过组合多个学习器来完成学习任务。

基分类器一般采用的是弱可学习分类器，通过集成方法，组合成一个强可学习分类器。

弱可学习：学习的正确率仅略优于随机猜测的多项式学习算法。

强可学习：正确率较高的多项式学习算法。

集成学习的泛化能力一般比单一的基分类器要好，因为大部分基分类器的分类错误的概率远低于单一基分类器的。

集成方法主要包括Bagging和Boosting两种方法，Bagging和Boosting都是将已有的分类或回归算法通过一定方式组合起来，形成一个性能更加强大的分类器，更准确的说这是一种分类算法的组装方法，即将弱分类器组装成强分类器的方法。

## **Bagging**

自举汇聚法，也称为bagging方法。

Bagging对训练数据采用自举采样，即有放回地采样数据，主要思想是：

从原始样本集中抽取训练集。每轮从原始样本集中使用Bootstraping的方法抽取n个训练样本。共进行k轮抽取，得到k个训练集。

在训练集中，有些样本可能被多次抽取到，而有些样本可能一次都没有被抽中。

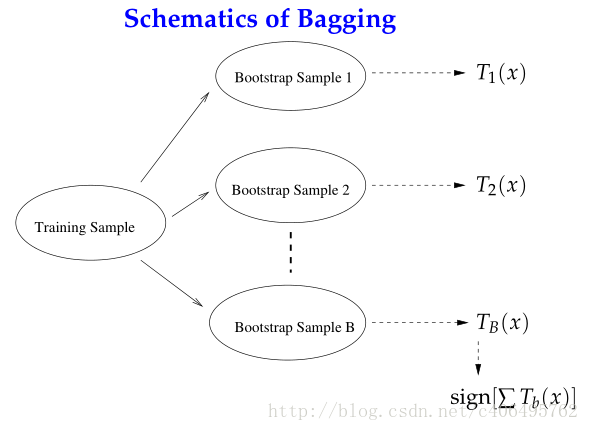
k个训练集之间是相互独立的。

每次使用一个训练集得到一个模型，k个训练集共得到k个模型。

（注：并没有具体的分类算法或回归方法，可以根据具体问题采用不同的分类或回归方法，如决策树等）

对分类问题：将上步得到的k个模型采用投票的方式得到分类结果

对回归问题：计算上述模型的均值作为最后的结果。



## 

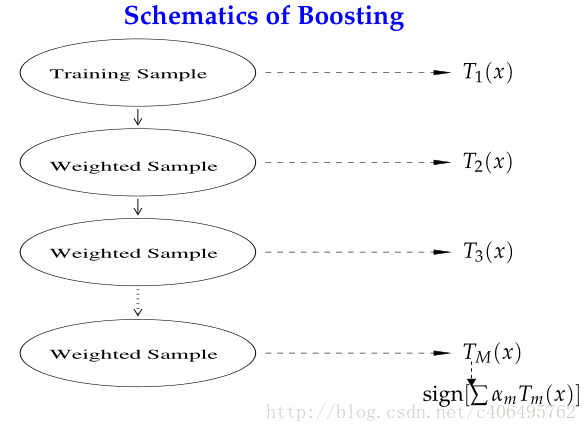
## **Boosting**

Boosting是一种与Bagging很类似的技术。

Boosting的思路则是采用重赋权法迭代地训练基分类器，主要思想是：

每一轮的训练数据样本赋予一个权重，并且每一轮样本的权值分布依赖上一轮的分类结果。

基分类器之间采用序列式的线性加权方式进行组合。



## 

## **Bagging、Boosting二者之间的区别**

从样本选择上：

Bagging：训练集是在原始集中有放回选取的，从原始集中选出的各轮训练集之间是独立的。

Boosting：每一轮的训练集不变，只是训练集中每个样例在分类器中的权重发生变化。而权值是根据上一轮的分类结果进行调整。

从样例权重上：

Bagging：使用均匀取样，每个样例的权重相等。

Boosting：根据错误率不断调整样例的权值，错误率越大则权重越大。

从预测函数上：

Bagging：所有预测函数的权重相等。

Boosting：每个弱分类器都有相应的权重，对于分类误差小的分类器会有更大的权重。

从并行计算上：

Bagging：各个预测函数可以并行生成。

Boosting：各个预测函数只能顺序生成，因为后一个模型参数需要前一轮模型的结果。

## 

## **总结**

这两种方法都是把若干个分类器整合为一个分类器的方法，只是整合的方式不一样，最终得到不一样的效果，将不同的分类算法套入到此类算法框架中一定程度上会提高了原单一分类器的分类效果，但是也增大了计算量。

下面是将决策树与这些算法框架进行结合所得到的新的算法：

Bagging + 决策树 = 随机森林

AdaBoost + 决策树 = 提升树

Gradient Boosting + 决策树 = GBDT

集成方法众多，这次要学的方法是Boosting方法中的一种最流行的版本，即AdaBoost。

# **AdaBoost**

AdaBoost算法是基于Boosting思想的机器学习算法，AdaBoost是adaptive boosting（的缩写，其运行过程如下：

**第一步：计算样本权重**

训练数据中的每个样本，赋予其权重，即样本权重，用向量D表示，这些权重都初始化成相等值。

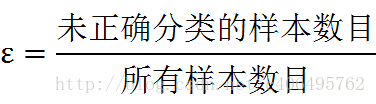
假设有n个样本的训练集：

IMG_256

设定每个样本的权重都是相等的，即1/n。

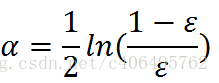
**第二部：计算错误率**

利用第一个弱学习算法h1对其进行学习，学习完成后进行错误率ε的统计。



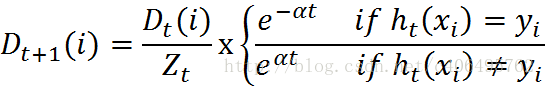
**第三步：计算弱学习算法权重**

弱学习算法也有一个权重，用向量α表示，利用错误率计算权重α。



**第四步：更新样本权重**

在第一次学习完成后，需要重新调整样本的权重，以使得在第一分类中被错分的样本的权重，在接下来的学习中可以重点对其进行学习。

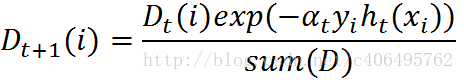


其中，h\_t(x\_i) = y\_i表示对第i个样本训练正确，不等于则表示分类错误。

Z\_t是一个归一化因子：

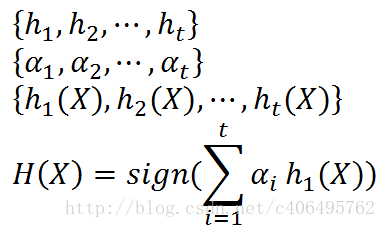
IMG_260

这个公式可以继续化简，将两个公式进行合并，化简如下：



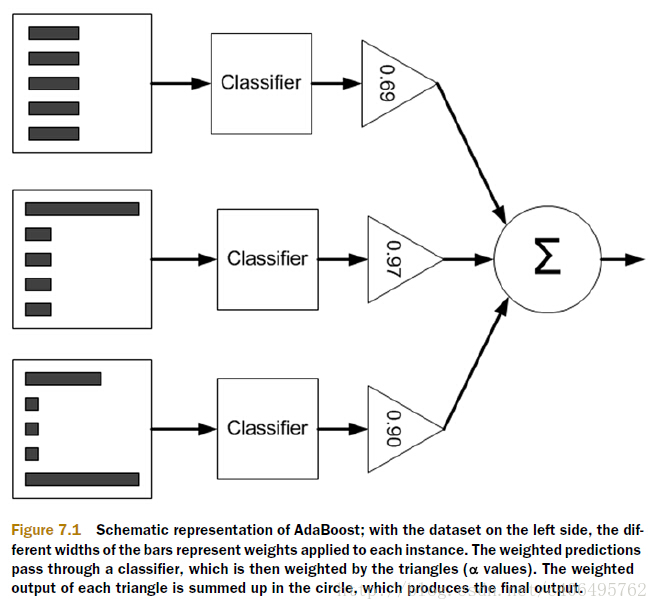
**第五步：AdaBoost算法**

重复进行学习，这样经过t轮的学习后，就会得到t个弱学习算法、权重、弱分类器的输出以及最终的AdaBoost算法的输出，分别如下：

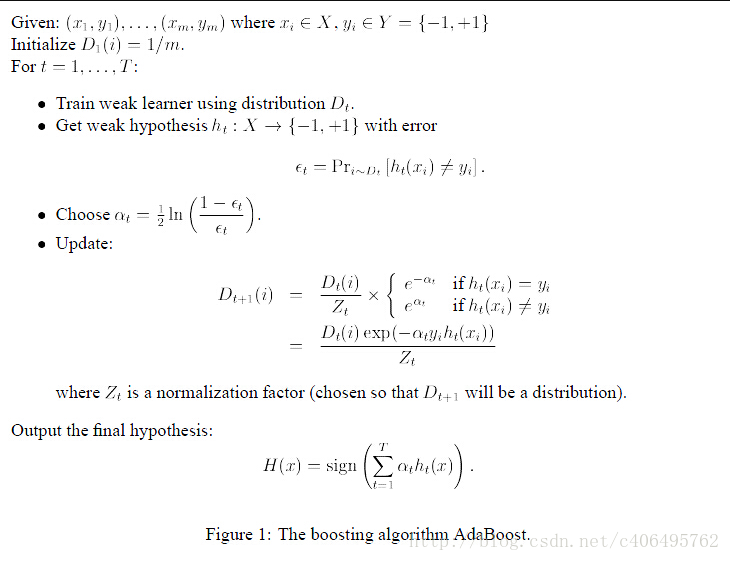


其中，sign(x)是符号函数。

具体过程如下所示：



AdaBoost算法总结如下：



**基于单层决策树构建弱分类器**

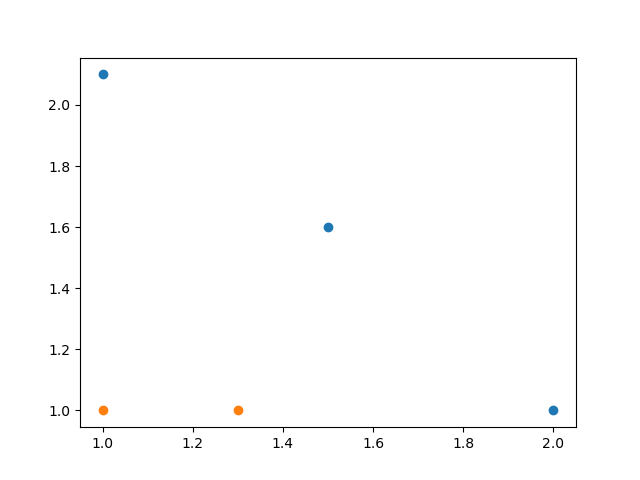
建立AdaBoost算法之前，必须先建立弱分类器，并保存样本的权重。

弱分类器使用单层决策树，也称决策树桩，它是一种简单的决策树，通过给定的阈值，进行分类。

**AdaBoost.py**

**数据集可视化**

为了训练单层决策树，需要创建一个训练集，编写代码如下：

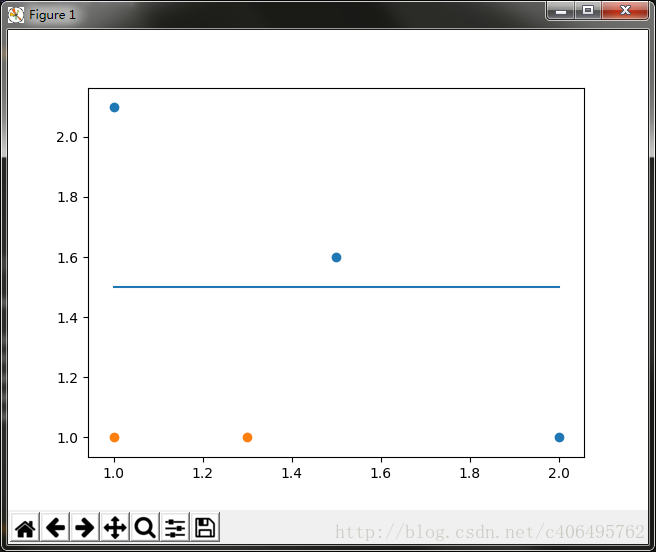
如图所示，如果我们想要从某个坐标轴上选择一个值（即选择一条与坐标轴平行的直线）来将所有的蓝色圆点和橘色圆点分开，这显然是不可能的。这就是单层决策树难以处理的问题。

通过使用多颗单层决策树，可以构建出一个能够对该数据集完全正确分类的分类器。

**AdaBoost-1.py**

**构建单层决策树**

设置一个分类阈值，比如横向切分，如下图所示：



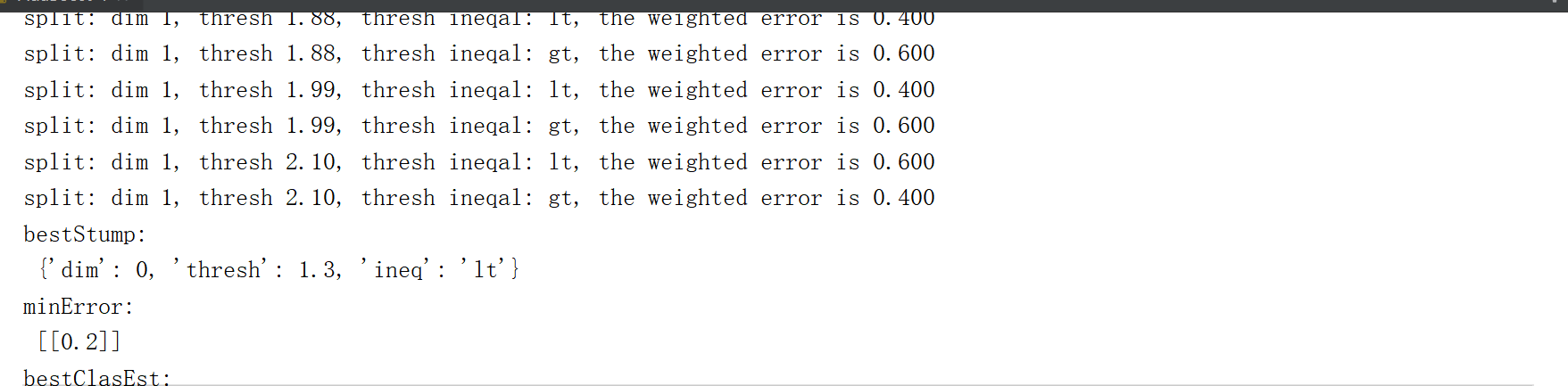
蓝横线上边的是一个类别，蓝横线下边是一个类别。

此时有一个蓝点分类错误，计算此时的分类误差，误差为1/5 = 0.2。

这个横线与坐标轴的y轴的交点，就是我们设置的阈值，通过不断改变阈值的大小，找到使单层决策树的分类误差最小的阈值。

同理，竖线也是如此，找到最佳分类的阈值，就找到了最佳单层决策树。

代码输出如下：





通过遍历，改变不同的阈值，计算最终的分类误差，找到分类误差最小的分类方式，即为我们要找的最佳单层决策树。

lt表示less than，表示分类方式，对于小于阈值的样本点赋值为-1.

gt表示greater than，也是表示分类方式，对于大于阈值的样本点赋值为-1。

经过遍历，我们找到了训练好的最佳单层决策树的最小分类误差为0.2，就是对于该数据集，无论用什么样的单层决策树，分类误差最小就是0.2。这就是我们训练好的弱分类器。

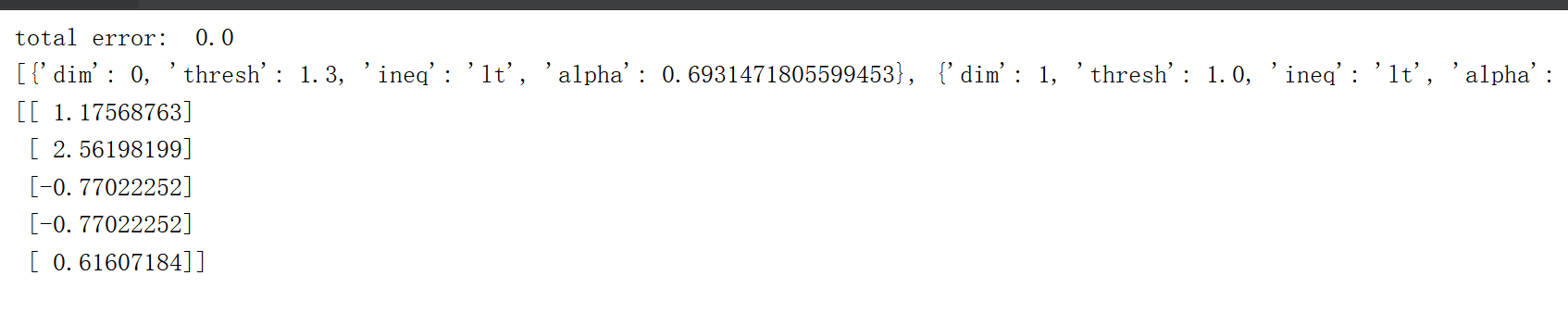
接下来，使用AdaBoost算法提升分类器性能，将分类误差缩短到0，看下AdaBoost算法是如何实现的。

**AdaBoost-2.py**

**使用AdaBoost提升分类器性能**

根据之前介绍的AdaBoost算法实现过程，使用AdaBoost算法提升分类器性能。

代码输出如下：



在第一轮迭代中，D中的所有值都相等。只有第一个数据点被错分了。

因此在第二轮迭代中，D向量给第一个数据点0.5的权重。这就可以通过变量aggClassEst的符号来了解总的类别。第二次迭代之后，我们发现第一个数据点已经正确分类了，但此时最后一个数据点却是错分了。D向量中的最后一个元素变为0.5，而D向量中的其他值都变得非常小。

最后，第三次迭代之后aggClassEst所有值的符号和真是类别标签都完全吻合，那么训练错误率为0，程序终止运行。

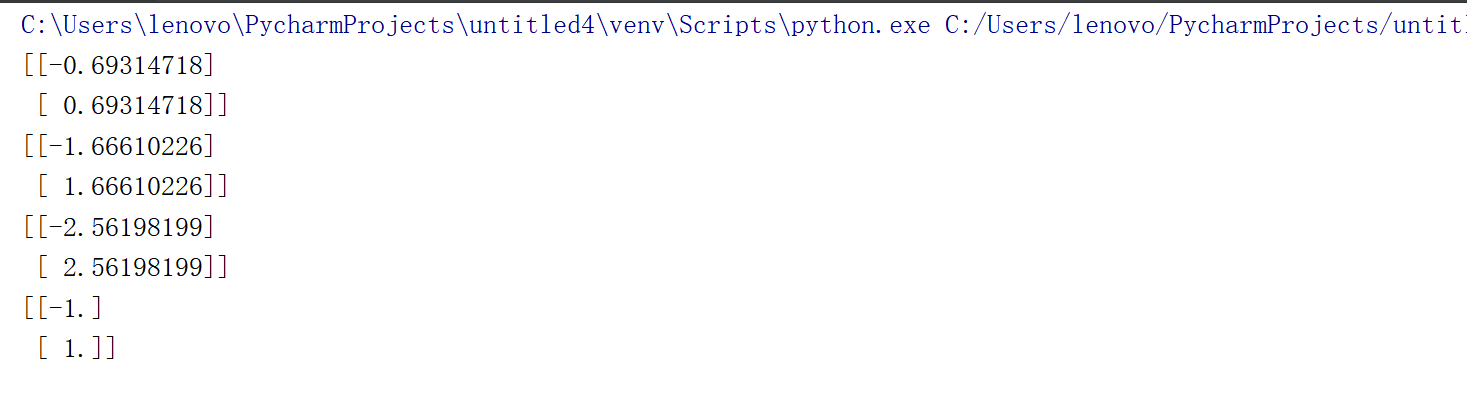
最后训练结果包含了三个弱分类器，其中包含了分类所需要的所有信息。

一共迭代了3次，所以训练了3个弱分类器构成一个使用AdaBoost算法优化过的分类器，分类器的错误率为0。

一旦拥有了多个弱分类器以及其对应的alpha值，进行测试就变得想当容易了。

**AdaBoost-3.py**

代码输出如下：



在之前代码的基础上，添加了一个可以遍历所有训练得到的弱分类器的函数---adaClassify()函数。

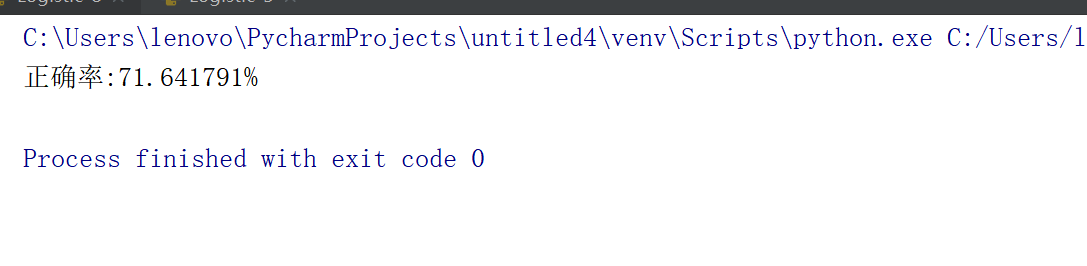
利用单层决策树，输出的类别估计值乘以该单层决策树的分类器权重alpha，然后累加到aggClassEst上，最后通过sign函数最终的结果。

可以看到，分类没有问题，(5,5)属于正类，(0,0)属于负类。

**在一个难数据集上应用AdaBoost**

在前面我们使用Logistic回归方法训练马疝病数据集，预测病马死亡率。

当时的训练结果如下图所示：



这个是使用Sklearn的LogisticRegression()训练的分类器，可以看到，正确率约为71.641791%，也就是说错误率约为28%。

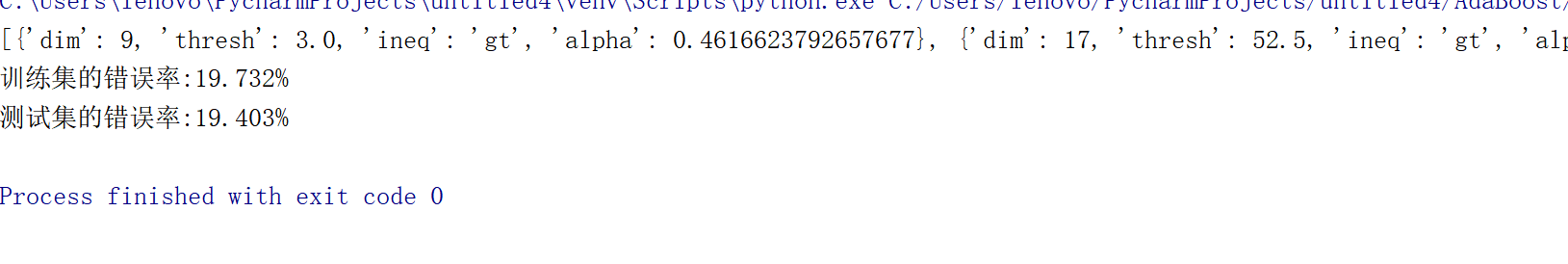
错误率还是挺高的，现在我们使用AdaBoost算法，训练出一个更强的分类器。

我们使用的数据集有所变化，之前的标签是0和1，现在将标签改为+1和-1，其他数据不变。

**AdaBoost-4.py**

使用Python写的AbaBoost算法进行训练，添加loadDataSet函数用于加载数据集。

代码输出如下：



这里输出了AdaBoost算法训练好的分类器的组合，我们只迭代了40次，也就是训练了40个弱分类器。

最终，训练集的错误率为19.732%，测试集的错误率为19.403%，可以看到相对于Sklearn的罗辑回归方法，错误率降低了很多。

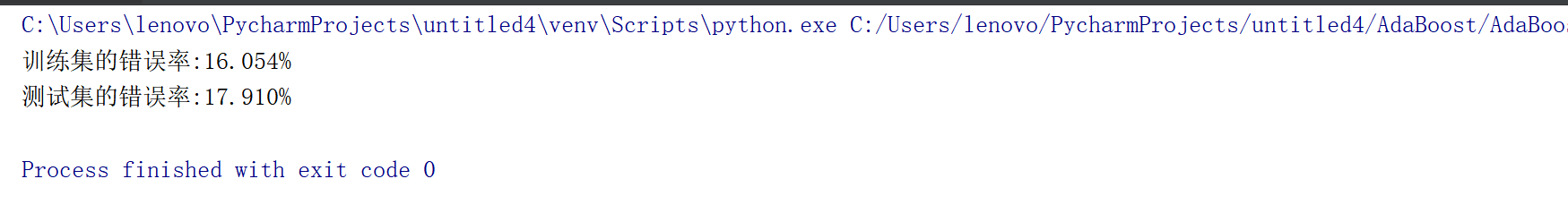
这个仅仅是我们训练40个弱分类器的结果，如果训练更多弱分类器，效果会更好。

但是当弱分类器数量过多的时候，你会发现训练集错误率降低很多，但是测试集错误率提升了很多，这种现象就是过拟合(overfitting)。

分类器对训练集的拟合效果好，但是缺失了普适性，只对训练集的分类效果好，这是我们不希望看到的。

训练多个分类器

代码输出如下：

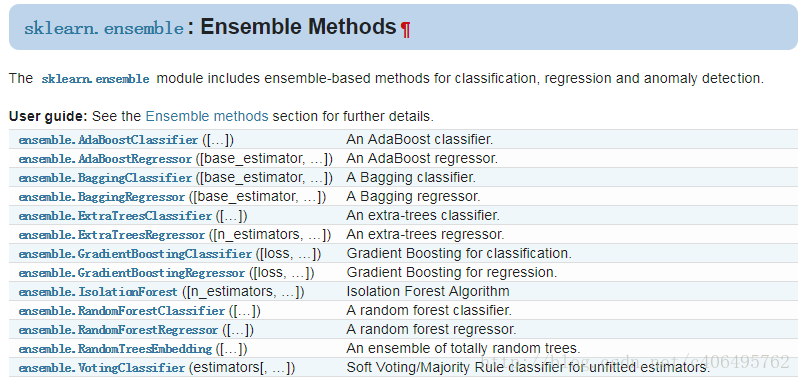


**使用Sklearn的AdaBoost**

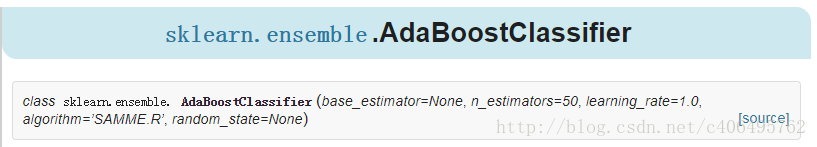
**AdaBoost-5.py**

sklearn.ensemble模块提供了很多集成方法，AdaBoost、Bagging、随机森林等。

我们使用的是AdaBoostClassifier方法。



让我们先看下AdaBoostClassifier这个函数，一共有5个参数：



参数说明如下：

**base\_estimator**：可选参数，默认为DecisionTreeClassifier。

理论上可以选择任何一个分类或者回归学习器，不过需要支持样本权重。我们常用的一般是CART决策树或者神经网络MLP。

默认是决策树，即AdaBoostClassifier默认使用CART分类树DecisionTreeClassifier，而AdaBoostRegressor默认使用CART回归树DecisionTreeRegressor。

**algorithm：**可选参数，默认为SAMME.R。

scikit-learn实现了两种Adaboost分类算法，SAMME和SAMME.R。两者的主要区别是弱学习器权重的度量，SAMME使用对样本集**分类效果**作为弱学习器权重，而SAMME.R使用了对样本集分类的**预测概率大小**来作为弱学习器权重。

由于SAMME.R使用了概率度量的连续值，迭代一般比SAMME快，因此AdaBoostClassifier的默认算法algorithm的值也是SAMME.R。

但是要注意的是使用了SAMME.R，则弱分类学习器参数base\_estimator必须限制使用支持概率预测的分类器。SAMME算法则没有这个限制。

**n\_estimators：**整数型，可选参数，默认为50。

弱学习器的最大迭代次数，或者说最大的弱学习器的个数。

一般来说n\_estimators太小，容易欠拟合，但n\_estimators太大，又容易过拟合，一般选择一个适中的数值。默认是50。

在实际调参的过程中，常常将n\_estimators和参数learning\_rate一起考虑。

**learning\_rate：**浮点型，可选参数，默认为1.0。

每个弱学习器的权重缩减系数，取值范围为0到1，对于同样的训练集拟合效果，较小的意味着需要更多的弱学习器的迭代次数。

通常用步长和迭代最大次数一起来决定算法的拟合效果。

所以这两个参数n\_estimators和learning\_rate要一起调参。一般来说，可以从一个小一点的开始调参，默认是1。

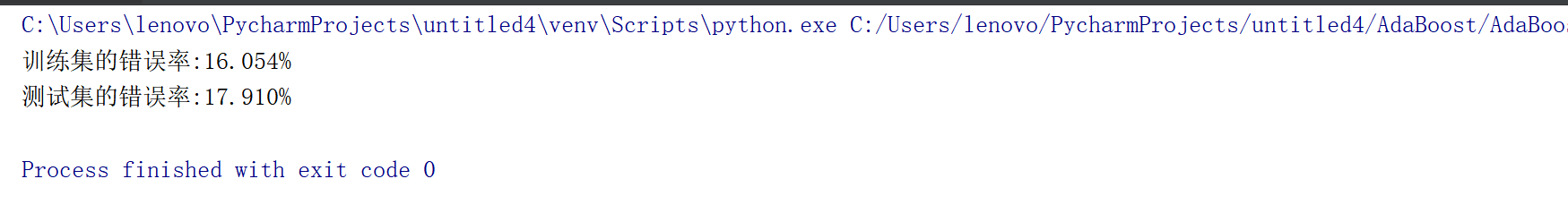
**random\_state：**整数型，可选参数，默认为None。

如果是RandomState的实例，random\_state是随机数生成器;

如果是None，则随机数生成器是由np.random使用的RandomState实例。

了解这些，我们就可以开始编写代码了。

代码输出如下：

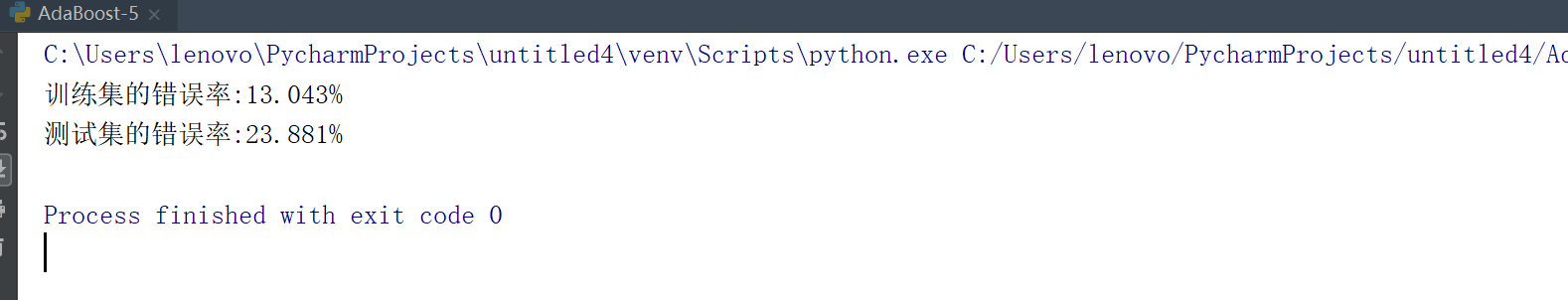


使用DecisionTreeClassifier作为使用的弱分类器，使用AdaBoost算法训练分类器。可以看到训练集的错误率为16.054%，测试集的错误率为：17.910%。

更改n\_estimators参数，发现跟上面更改迭代次数的效果是一样的。n\_enstimators参数过大，会导致过拟合。

当n\_enstimators=20时

代码输出如下：



**分类器性能评价**

在之前,我们学习的很多分类器，都是假设所有类别的分类代价是一样的。

很多时候，不同类别的分类代价并不相等，这就是非均衡分类问题。

考察一种新的分类器性能度量方法，不再是简单的通过错误率进行评价，并且通过图像技术来对上述非均衡问题下不同分类器性能进行可视化处理。

**分类器性能度量指标**

在之前，我们都是基于错误率来衡量分类器任务的成功程度的。

错误率指的是在所有测试样本中错分的样本比例。实际上，这样的度量错误掩盖了样例如何被错分的事实。

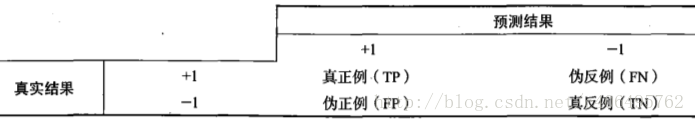
在机器学习中，有一个普遍适用的称为混淆矩阵的工具，它可以帮助人们更好地了解分类中的错误。

利用混淆矩阵就可以更好地理解分类中的错误了。

如果矩阵中的飞对角元素均为0，就会得到一个完美的分类器。

我们考虑一个混淆矩阵，这次的矩阵针对一个简单的二类问题。

混淆矩阵如下图所示：



可以看到，在这个二分类问题中，如果对一个正例正确地判为正例，那么就可以认为产生了一个真正例。

如果对一个反例正确地判为反例，则认为产生了一个真反例。

如果对一个正例错误地判为反例，那么就可以认为产生了一个伪反例。

如果对一个反例错误地判为正例，则认为产生了一个伪正例。

在分类中，当某个类别的重要性高于其他类别时，我们就可以来利用上述定义来定义出多个比错误率更好的指标。

从混淆矩阵中，也可以衍生出**各种评价指标**。

各个指标的定义及含义如下：

1）Accuracy

模型的精度，即模型预测正确的个数/样本的总个数

IMG_259

一般情况下，模型的精度越高，说明模型的效果越好。

2）Positive predictive value

正确率，阳性预测值，在模型预测为正类的样本中，真正的正样本所占的比例。

IMG_260

一般情况下，正确率越高，说明模型的效果越好。

3）False discovery rate

伪发现率，也是错误发现率，表示在模型预测为正类的样本中，真正的负类的样本所占的比例。

IMG_261

一般情况下，错误发现率越小，说明模型的效果越好。

4）False omission rate

错误遗漏率，表示在模型预测为负类的样本中，真正的正类所占的比例。即评价模型"遗漏"掉的正类的多少。

IMG_262

5）Negative predictive value

阴性预测值，在模型预测为负类的样本中，真正为负类的样本所占的比例。

IMG_263

一般情况下，NPV越高，说明的模型的效果越好。

6）True positive rate

召回率，真正类率，表示的是，模型预测为正类的样本的数量，占总的正类样本数量的比值。

IMG_264

一般情况下，Recall越高，说明有更多的正类样本被模型预测正确，模型的效果越好。

1. False positive rate

假正率，表示的是，模型预测为正类的样本中，占模型负类样本数量的比值。

IMG_265

一般情况下，假正类率越低，说明模型的效果越好。

8）False negative rate

假负类率，缺失率，模型预测为负类的样本中，是正类的数量，占真实正类样本的比值。

IMG_266

缺失值越小，说明模型的效果越好。

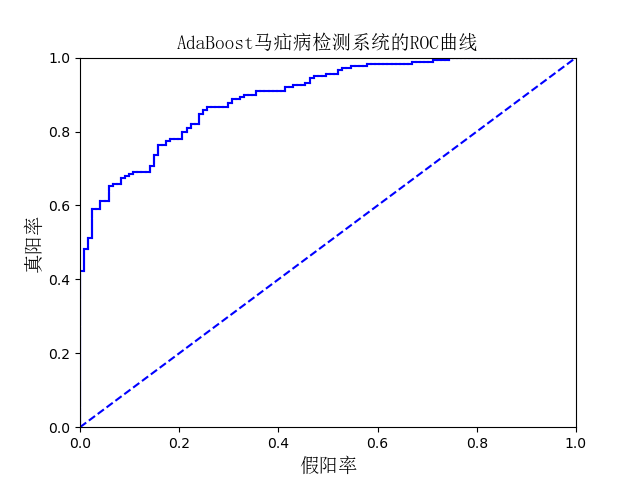
我们可以很容易构造一个高正确率或高召回率的分类器，但是很难同时保证两者成立。

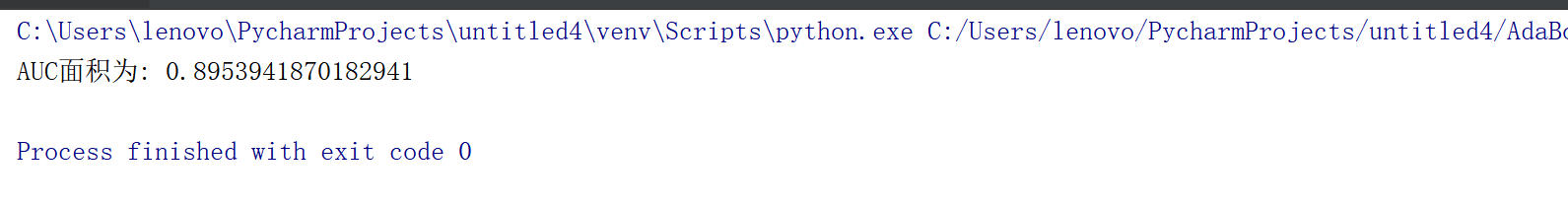
如果将任何样本都判为正例，那么召回率达到百分之百而此时正确率很低。

**AdaBoost-6.py**

除了上述的评价指标，另一个用于度量分类中的非均衡的工具是ROC曲线。

代码输出如下：





有两个输出结果，一个是AUC面积，另一个ROC曲线图。

我们先学习ROC，ROC图中的横坐标是伪正例的比例，而纵坐标是真正例的比例。ROC曲线给出的是当阈值变化时假阳率和真阳率的变化情况。

左下角的点对应的将所有样例判为反例的情况，而右上角的点对应的则是将所有样例判为正例的情况。虚线给出的是随机猜测的结果曲线。

在理想的情况下，最佳的分类器应该尽可能地处于左上角，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。

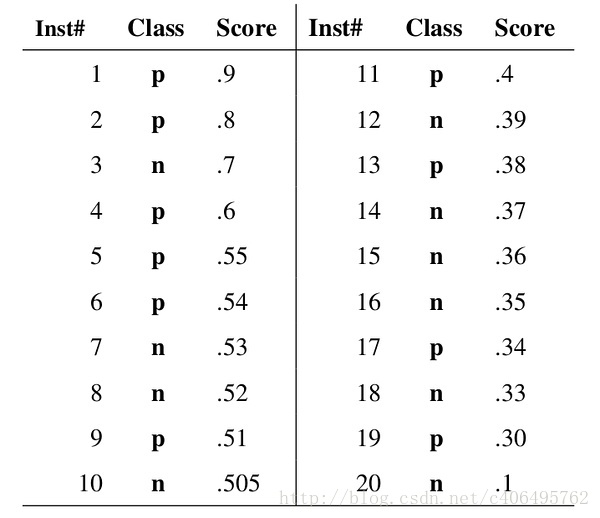
对不同的ROC曲线进行比较的一个指标是曲线下的面积----AUC。

AUC给出的是分类器的平均性能值，当然它并不能完全代替对整条曲线的观察。一个完美分类器的ACU为1.0，而随机猜测的AUC则为0.5。

**那么，我们怎么画ROC曲线呢？**

对于分类器而言，都有概率输出的功能。

现在假设我们已经得到了所有样本的概率输出，根据每个测试样本属于正样本的概率值从大到小排序。如下图所示：



图中共有20个测试样本，“Inst”一栏表示样本编号，“Class”一栏表示每个测试样本真正的标签（p表示正样本，n表示负样本），“Score”表示每个测试样本属于正样本的概率。

其中，一共10个正样本，10个负样本。

接下来，我们从高到低，依次将“Score”值作为阈值，当测试样本属于正样本的概率大于或等于时，认为它为正样本，否则为负样本。

举例个例子：

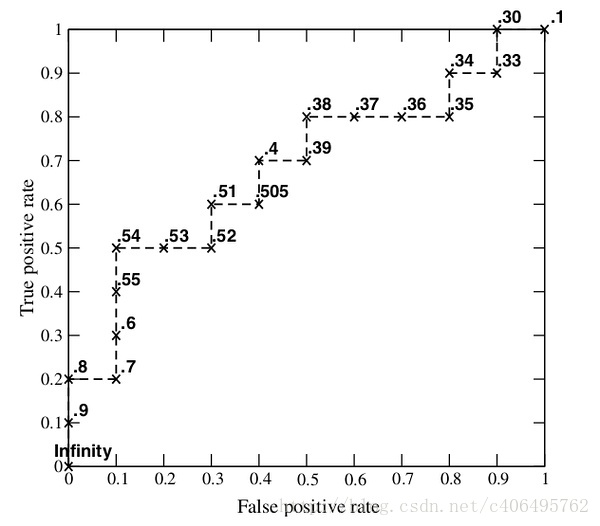
当阈值为0.9时，样本1的“Score”值为0.9，大于等于阈值，那么样本1为正类，其余为负类，则TPR=TP/（TP+FN）=1/10=0.1，FPR=FP/（TP+FN）=0/10=0；

当阈值为0.6时，样本1，2，3，4的"Score"值大于等于阈值，那么样本1，2，3，4为正类，其余为负类，则TPR=3/10=0.3，FPR=1/10=0.1；

以此类推

当阈值为0.1时，所有样本的"Score"值大于等于阈值，那么所有样本均为正类，则TPR=10/10=1，FPR=10/10=1。

将它们的结果画出来，就构成了下图：



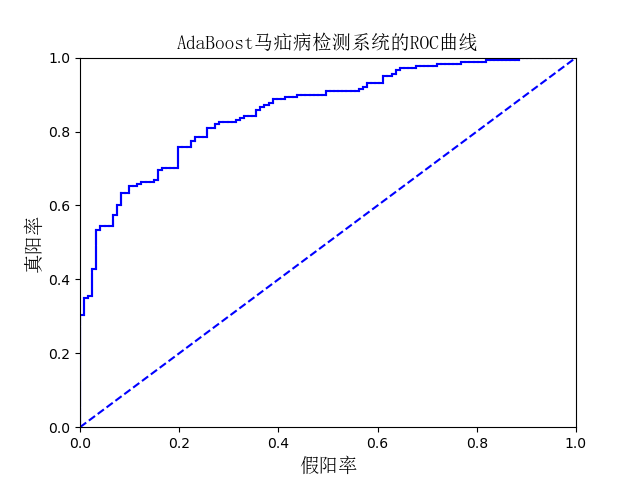
根据图像，我们发现了这样一个规律：多了一个TP，向Y轴移动一步，多了一个FP，向X轴移动一步。

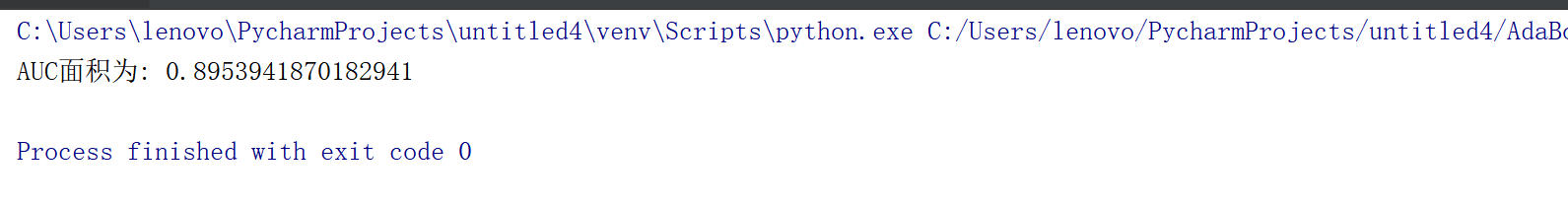
**AUC是如何计算的呢？**

很简单，这些小矩形的宽度都是固定的xStep，因此先对所有矩形的高度进行累加，即ySum，最后面积就是ySum\*xStep。

上面的ROC曲线绘制结果是在10个弱分类器下，AdaBoost算法性能的结果。

将迭代次数改为50，也就是训练50个弱分类器，看下ROC曲线和AUC的变化：





ROC曲线往左上角更靠拢了，并且AUC值增加了。也就表明，分类器效果更佳。

这就是ROC和AUC对与分类器评价指标，ROC越靠拢于左上角，分类器性能越好。同理AUC越接近于1，分类器性能越好。

**总结**

AdaBoost的优缺点：

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。

缺点：对离群点敏感。