菊安酱的机器学习第6期

菊安酱的直播间：https://live.bilibili.com/14988341

每周一晚8:00 菊安酱和你不见不散哦~(^o^)/~

更新日期：2018-12-10

作者：菊安酱

课件内容说明：

- 本文为作者原创，转载请注明作者和出处
- 如果想获得此课件及录播视频，可扫描左边二维码，回复"k"进群
- 如果想获得2小时完整版视频，可扫描右边二维码或点击如下链接
- 若有任何疑问，请给作者留言。

交流群二维码

完整版视频及课件

直播视频及课件：http://www.peixun.net/view/1278.html

完整版视频及课件：http://edu.cda.cn/course/966

12期完整版课纲

直播时间：每周一晚8:00

直播内容：

时间	期数	算法
2018/11/05	第1期	k-近邻算法
2018/11/12	第2期	决策树
2018/11/19	第3期	朴素贝叶斯
2018/11/26	第4期	Logistic回归
2018/12/03	第5期	支持向量机
2018/12/10	第6期	AdaBoost 算法
2018/12/17	第7期	线性回归
2018/12/24	第8期	树回归
2018/12/31	第9期	K-均值聚类算法
2019/01/07	第10期	Apriori 算法
2019/01/14	第11期	FP-growth 算法
2019/01/21	第12期	奇异值分解SVD

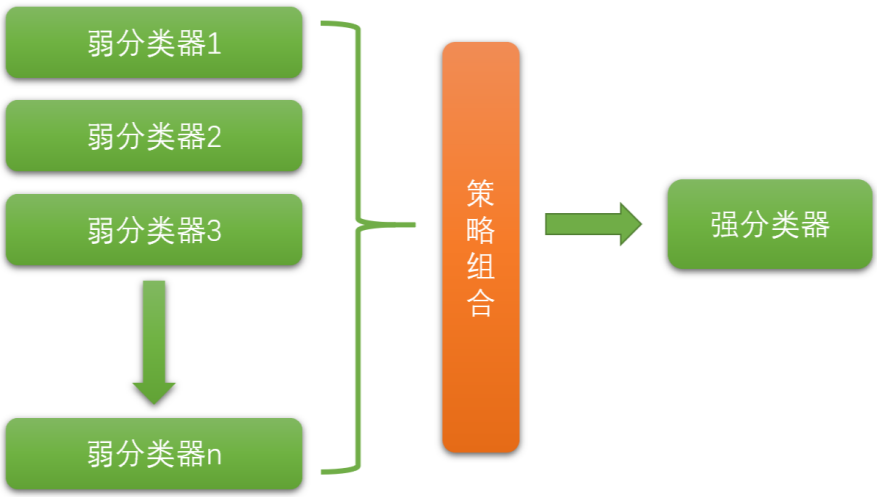
Adaboost 算法

菊安酱的机器学习第6期
12期完整版课纲
Adaboost 算法
一、集成学习概述
    1. 集成学习算法定义
    2. bagging（装袋）
    3. boosting（提升）
    4. 结合策略
        4.1 平均法
        4.2 投票法
        4.3 学习法
二、Adaboost 算法
    1. 计算样本权重
    2. 计算错误率
    3. 计算弱分类器权重
    4. 调整权重值
    5. 最终强分类器结果
三、基于单层决策树构建弱分类器
    1. 构建简单数据集
    2. 构建单层决策树
四、完整AdaBoost算法的实现
五、基于AdaBoost的分类
六、案例：在病马数据集上应用AdaBoost
    1. 导入数据集
    2. 构建分类函数
七、过拟合和欠拟合
八、分类器的衡量指标
    1. 混淆矩阵
    2. ROC曲线
    3. AUC面积
九、样本不均衡问题
    1. 样本层面的处理
    2. 算法层面的处理
    3. 改变样本的权重
十、实例：泰坦尼克幸存者预测
    1. 导入数据集并进行预处理
    2. 划分训练集和测试集
    3. 建立AdaBoost分类模型
    4. 其他集成算法结果

一、集成学习概述

1. 集成学习算法定义

集成学习（Ensemble learning）就是将若干个弱分类器通过一定的策略组合之后产生一个强分类器。弱分类器（Weak Classifier）指的就是那些分类准确率只比随机猜测略好一点的分类器，而强分类器（ Strong Classifier）的分类准确率会高很多。这里的"强"&"弱"是相对的。某些书中也会把弱分类器称为“**基分类器**”。



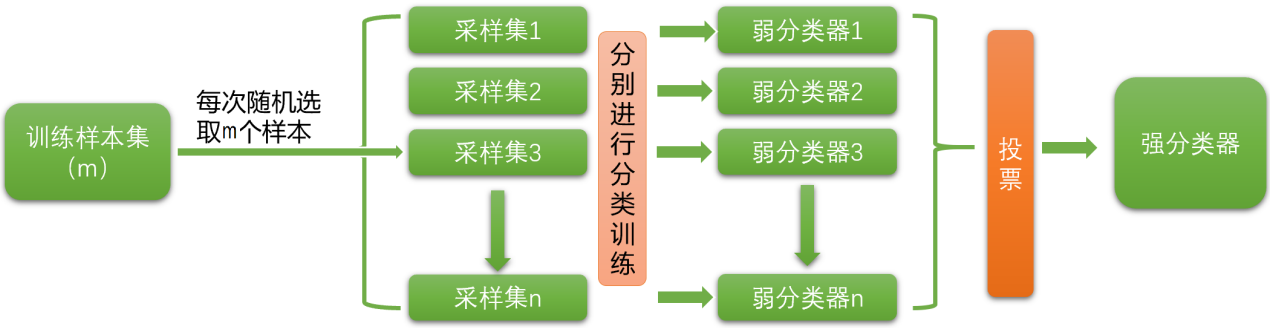
目前集成学习算法的流派主要有两种：

- bagging
- boosting

2. bagging（装袋）

装袋（bagging）又称自主聚集（bootstrap aggregating），是一种根据均匀概率分布从数据集中重复抽样（有放回的）的技术。每个新数据集和原始数据集的大小相等。由于新数据集中的每个样本都是从原始数据集中**有放回的随机抽样**出来的，所以新数据集中可能有重复的值，而原始数据集中的某些样本可能根本就没出现在新数据集中。

bagging方法的流程，如下图所示：



bagging图示

有放回的随机抽样：自主采样法（Bootstap sampling），也就是说对于m个样本的原始数据集，每次随机选取一个样本放入采样集，然后把这个样本重新放回原数据集中，然后再进行下一个样本的随机抽样，直到一个采样集中的数量达到m，这样一个采样集就构建好了，然后我们可以重复这个过程，生成n个这样的采样集。也就是说，最后形成的采样集，每个采样集中的样本可能是重复的，也可能原数据集中的某些样本根本就没抽到，并且每个采样集中的样本分布可能都不一样。

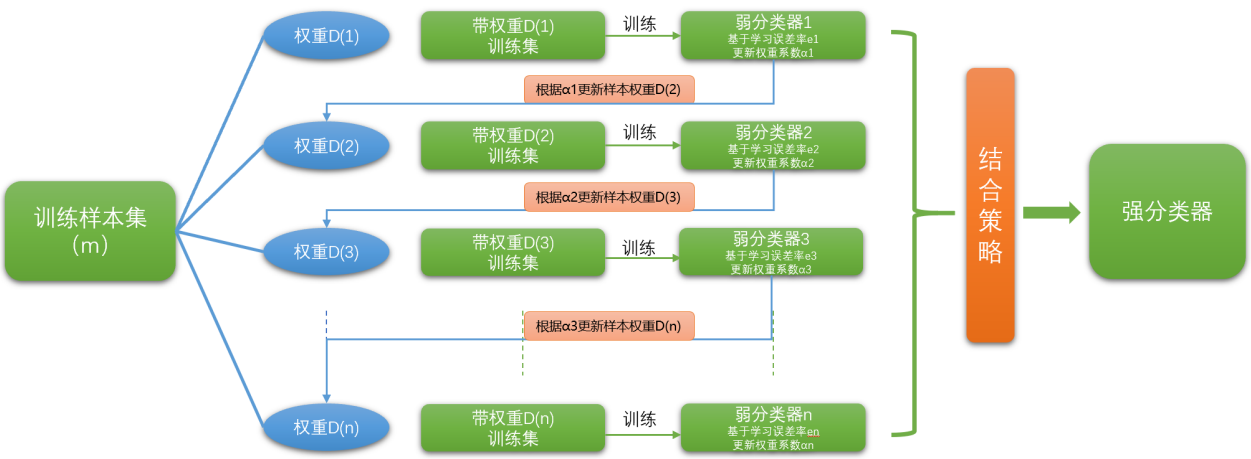
根据有放回的随机抽样构造的n个采样集，我们就可以对它们分别进行训练，得到n个弱分类器，然后根据每个弱分类器返回的结果，我们可以采用一定的**组合策略**得到我们最后需要的强分类器。

bagging方法的代表算法是**随机森林**，准确的来说，随机森林是bagging的一个特化进阶版，所谓的特化是因为随机森林的弱学习器都是决策树。所谓的进阶是随机森林在bagging的样本随机采样基础上，又加上了特征的随机选择，其基本思想没有脱离bagging的范畴。

3. boosting（提升）

boosting是一个迭代的过程，用来自适应地改变训练样本的分布，使得弱分类器聚焦到那些很难分类的样本上。它的做法是给每一个训练样本赋予一个权重，在每一轮训练结束时自动地调整权重。

boosting方法的流程，如下图所示：



boosting图示

boosting方法的代表算法有**Adaboost**、**GBDT**、**XGBoost**算法

4. 结合策略

这里列举出三种结合策略

4.1 平均法

对于数值类的回归预测问题，通常使用的结合策略是平均法，也就是说，对于若干个弱学习器的输出进行平均得到最终的预测输出。

假设我们最终得到的n个弱分类器为$\{h_1, h_2, \dots, h_n\}$

最简单的平均是算术平均，也就是说最终预测是

$$H(x) = \frac{1}{n}\sum_{1}^{n} h_i(x)$$

如果每个弱分类器有一个权重w，则最终预测是

$$H(x) = \frac{1}{n}\sum_{1}^{n} w_i h_i(x)$$

$$s.t.\ \ w_i \geq 0, \sum_{1}^{n} w_i = 1$$

4.2 投票法

对于分类问题的预测，我们通常使用的是投票法。假设我们的预测类别是$\{c_1, c_2, \dots c_K\}$,对于任意一个预测样本x，我们的n个弱学习器的预测结果分别是$(h_1(x), h_2(x) \dots h_n(x))$。

最简单的投票法是相对多数投票法，也就是我们常说的少数服从多数，也就是n个弱学习器的对样本x的预测结果中，数量最多的类别c_i为最终的分类类别。如果不止一个类别获得最高票，则随机选择一个做最终类别。

稍微复杂的投票法是绝对多数投票法，也就是我们常说的要票过半数。在相对多数投票法的基础上，不光要求获得最高票，还要求票过半数。否则会拒绝预测。

更加复杂的是加权投票法，和加权平均法一样，每个弱学习器的分类票数要乘以一个权重，最终将各个类别的加权票数求和，最大的值对应的类别为最终类别。

4.3 学习法

前两种方法都是对弱学习器的结果做平均或者投票，相对比较简单，但是可能学习误差较大，于是就有了学习法这种方法，对于学习法，代表方法是stacking，当使用stacking的结合策略时， 我们不是对弱学习器的结果做简单的逻辑处理，而是再加上一层学习器，也就是说，我们将训练集弱学习器的学习结果作为输入，将训练集的输出作为输出，重新训练一个学习器来得到最终结果。

在这种情况下，我们将弱学习器称为初级学习器，将用于结合的学习器称为次级学习器。对于测试集，我们首先用初级学习器预测一次，得到次级学习器的输入样本，再用次级学习器预测一次，得到最终的预测结果。

二、Adaboost 算法

Adaboost是adaptive boosting(自适应boosting)的缩写。

算法步骤如下：

假设我们有m个样本的训练集，标签为 $y_i \in \{-1, +1\}$ ，我们的n个弱学习器的预测结果分别是 $(h_1(x), h_2(x) \dots h_n(x))$ ，t=1,2,3,...,n

1. 计算样本权重

赋予训练集中每个样本一个权重，构成权重向量D，将权重向量D初始化相等值。

设定我们有m个样本，每个样本的权重都相等，则权重为$\frac{1}{m}$。

2. 计算错误率

在训练集上训练出一个弱分类器，并计算分类器的错误率：

$$\epsilon = \frac{分错的数量}{样本总数}$$

3. 计算弱分类器权重

为当前分类器赋予权重值alpha，则alpha计算公式为：

$$\alpha = \frac{1}{2} ln(\frac{1-\epsilon}{\epsilon})$$

4. 调整权重值

根据上一次训练结果，调整权重值（上一次分对的权重降低，分错的权重增加）

如果第i个样本被正确分类，则该样本权重更改为：

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha}}{Sum(D)}$$

如果第i个样本被分错，则该样本权重更改为：

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{\alpha}}{Sum(D)}$$

把上面两个公式汇整成一个：

$$\begin{aligned} D_i^{(t+1)} &= \frac{D_i^{(t)}}{Sum(D)} \begin{cases} e^{-\alpha}, if\ h_t(x_i) = y_i \\ e^{\alpha}, if\ h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_i^{(t)} exp(-\alpha y_i h_t(x_i))}{Sum(D)} \end{aligned}$$

之后，在同一数据集上再一次训练弱分类器，然后循环上述过程，直到训练错误率为0，或者弱分类器的数目达到指定值。

5. 最终强分类器结果

循环结束后，我们可以得到我们的强分类器的预测结果：

$$H(x) = sign(\sum_i^n (\alpha_t h_t(x)))$$

三、基于单层决策树构建弱分类器

单层决策树（decision stump）也称决策树桩，是一种简单的决策树。第2期我们已经讲过决策树的相关原理了，接下来我们一起来构建一个单层决策树，它仅仅基于单个特征来做决策。由于这棵树只有一次分裂过程，因此它实际上就是一个树桩。

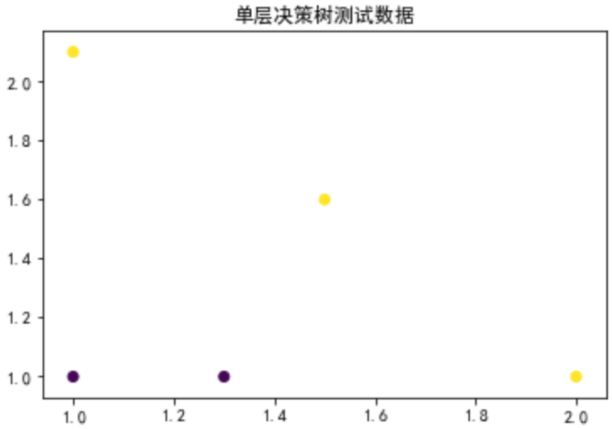
1. 构建简单数据集

我们先构建一个简单数据集来确保我们写出的函数能够正常运行。

```
import pandas as pd
import numpy as np

#获得特征矩阵和标签矩阵
def get_Mat(path):
    dataSet = pd.read_table(path,header = None)
    xMat = np.mat(dataSet.iloc[:,:-1].values)
    yMat = np.mat(dataSet.iloc[:,-1].values).T
    return xMat,yMat

xMat,yMat = get_Mat('simpdata.txt')
```

构建数据可视化函数，并运行查看数据分布

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['simhei']
%matplotlib inline

#数据集可视化函数
def showPlot(xMat,yMat):
    x=np.array(xMat[:,0])
    y=np.array(xMat[:,1])
    label = np.array(yMat)
    plt.scatter(x,y,c=label)
    plt.title('单层决策树测试数据')
    plt.show()

showPlot(xMat,yMat)
```



2. 构建单层决策树

我们会建立两个函数来实现我们的单层决策树：

第一个函数用来测试是否有某个值小于或者大于我们正在测试的阈值。

第二个函数稍微复杂一些，会在一个加权数据集中循环，并找到具有最低错误率的单层决策树

```
伪代码如下：
将最小错误率minE设为+∞
对数据集中每一个特征（第1层循环）：
    对每个步长（第2层循环）：
        对每个不等号（第3层循环）：
            建立一颗单层决策树并利用加权数据集对它进行预测
            如果错误率低于minE，则将当前单层决策树设为最佳单层决策树
返回最佳单层决策树
```

我们先来构建第一个函数：

```
"""
函数功能：单层决策树分类函数
参数说明：
    xMat：特征矩阵
    i：第i列，也就是第几个特征
    Q：阈值
    S：标志
返回：
    re：分类结果
"""
def ClassifyO(xMat,i,Q,S):
    re = np.ones((xMat.shape[0],1))                    #初始化re为1
    if S == 'lt':
        re[xMat[:,i] <= Q] = -1                        #如果小于阈值,则赋值为-1
    else:
        re[xMat[:,i] > Q] = -1                         #如果大于阈值,则赋值为-1
    return re
```

构建第二个函数寻找最佳单层决策树：

```
"""
函数功能：找到数据集上最佳的单层决策树
参数说明：
    xMat：特征矩阵
    yMat：标签矩阵
    D：样本权重
返回：
    bestStump：最佳单层决策树信息
    minE：最小误差
    bestClas：最佳的分类结果
"""
def get_Stump(xMat,yMat,D):
    m,n = xMat.shape                                    #m为样本个数，n为特征数
    Steps = 10                                          #初始化一个步数
    bestStump = {}                                      #用字典形式来储存树桩信息
    bestClas = np.mat(np.zeros((m,1)))                  #初始化分类结果为1
    minE = np.inf                                       #最小误差初始化为正无穷大
    for i in range(n):                                  #遍历所有特征
        Min = xMat[:,i].min()                           #找到特征中最小值
        Max = xMat[:,i].max()                           #找到特征中最大值
        stepSize = (Max - Min) / Steps                  #计算步长
        for j in range(-1, int(Steps)+1):
            for S in ['lt', 'gt']:                      #大于和小于的情况，均遍历。lt:less
than，gt:greater than
                Q = (Min + j * stepSize)                #计算阈值
                re = Classify0(xMat, i, Q, S)           #计算分类结果
                err = np.mat(np.ones((m,1)))            #初始化误差矩阵
                err[re == yMat] = 0                     #分类正确的,赋值为0
                eca = D.T * err                         #计算误差
                #print(f'切分特征: {i}, 阈值:{np.round(Q,2)}, 标志:{S}, 权重误差:
{np.round(eca,3)}')
                if eca < minE:                          #找到误差最小的分类方式
                    minE = eca
                    bestClas = re.copy()
                    bestStump['特征列'] = i
                    bestStump['阈值'] = Q
                    bestStump['标志'] = S
    return bestStump,minE,bestClas
```

测试函数并运行查看结果：

```
m = xMat.shape[0]
D = np.mat(np.ones((m, 1)) / m)  #初始化样本权重（每个样本权重相等）
bestStump,minE,bestClas= get_Stump(xMat,yMat,D)
```

四、完整AdaBoost算法的实现

完整版AdaBoost算法的实现伪代码如下：

```
对每一次迭代：
    利用bestStump()函数找到最佳的单层决策树
    将单层决策树加入到单层决策树数组
    计算分类器权重alpha
    更新样本权重向量D
    更新累积类别估计值
    如果错误率等于0，则退出循环
```

现在我们来用python代码来实现完整版AdaBoost算法

```
"""
函数功能：基于单层决策树的AdaBoost训练过程
参数说明：
    xMat：特征矩阵
    yMat：标签矩阵
    maxC：最大迭代次数
返回：
    weakClass：弱分类器信息
    aggClass：类别估计值（其实就是更改了标签的估计值）
"""
def Ada_train(xMat, yMat, maxC = 40):
    weakClass = []
    m = xMat.shape[0]
    D = np.mat(np.ones((m, 1)) / m)                          #初始化权重
    aggClass = np.mat(np.zeros((m,1)))
    for i in range(maxC):
        Stump, error, bestClas = get_Stump(xMat, yMat,D)     #构建单层决策树
        #print(f"D:{D.T}")
        alpha=float(0.5 * np.log((1 - error) / max(error, 1e-16))) #计算弱分类器权重alpha
        Stump['alpha'] = np.round(alpha,2)                   #存储弱学习算法权重,保留两位小数
        weakClass.append(Stump)                              #存储单层决策树
        #print("bestClas: ", bestClas.T)
        expon = np.multiply(-1 * alpha *yMat, bestClas)      #计算e的指数项
        D = np.multiply(D, np.exp(expon))
        D = D / D.sum()                                      #根据样本权重公式，更新样本权重
        aggClass += alpha * bestClas                         #更新累计类别估计值
        #print(f"aggClass: {aggClass.T}" )
        aggErr = np.multiply(np.sign(aggClass) != yMat, np.ones((m,1)))#计算误差
        errRate = aggErr.sum() / m
        #print(f"分类错误率: {errRate}")
        if errRate == 0: break                               #误差为0，退出循环
    return weakClass, aggClass
```

运行函数，查看结果：

```
weakClass, aggClass =Ada_train(xMat, yMat, maxC = 40)
weakClass
aggClass
```

五、基于AdaBoost的分类

这里我们使用弱分类器的加权求和来计算最后的结果。

```
"""
函数功能：AdaBoost分类函数
参数说明：
    data：待分类样例
    classifys:训练好的分类器
返回：
    分类结果
"""
def AdaClassify(data,weakClass):
    dataMat = np.mat(data)
    m = dataMat.shape[0]
    aggClass = np.mat(np.zeros((m,1)))
    for i in range(len(weakClass)):          #遍历所有分类器，进行分类
        classEst = Classify0(dataMat,
                             weakClass[i]['特征列'],
                             weakClass[i]['阈值'],
                             weakClass[i]['标志'])
        aggClass += weakClass[i]['alpha'] * classEst
        #print(aggClass)
    return np.sign(aggClass)
```

```
AdaClassify([0,0],weakClass)
```

六、案例：在病马数据集上应用AdaBoost

这里使用的数据集是在第4期讲解逻辑回归时使用的病马数据集，当时逻辑回归预测的结果如下：

In [46]:
```
def get_acc(train,test,alpha=0.001, maxCycles=5000):
    weights = SGD_LR(train,alpha=alpha,maxCycles=maxCycles)
    xMat = np.mat(test.iloc[:, :-1].values)
    xMat = regularize(xMat)
    result = []
    for inX in xMat:
        label = classify(inX,weights)
        result.append(label)
    retest=test.copy()
    retest['predict']=result
    acc = (retest.iloc[:,-1]==retest.iloc[:,-2]).mean()
    print(f'模型准确率为：{acc}')
    return retest
```

In [47]:
```
get_acc(train,test,alpha=0.001, maxCycles=5000)
```

```
模型准确率为：0.7611940298507462
```

这里使用的数据集简单进行了修改，把样本标签所有的0改成了-1。

1. 导入数据集

```
train = pd.read_table('horseColicTraining2.txt',header=None)
test = pd.read_table('horseColicTest2.txt',header=None)
```

2. 构建分类函数

```
def calAcc(maxC = 40):
    train_xMat,train_yMat = get_Mat('horseColicTraining2.txt')
    m=train_xMat.shape[0]
    weakClass, aggClass =Ada_train(train_xMat, train_yMat, maxC)
    yhat = AdaClassify(train_xMat,weakClass)
    train_re=0
    for i in range(m):
        if yhat[i]==train_yMat[i]:
            train_re+=1
    train_acc= train_re/m
    print(f'训练集准确率为{train_acc}')

    test_re=0
    test_xMat,test_yMat=get_Mat('horseColicTest2.txt')
    n=test_xMat.shape[0]
    yhat = AdaClassify(test_xMat,weakClass)
    for i in range(n):
        if yhat[i]==test_yMat[i]:
            test_re+=1
    test_acc=test_re/n
    print(f'测试集准确率为{test_acc}')
    return train_acc,test_acc
```

运行函数查看结果：

```
calAcc(maxC = 40)
```

我们仅用40个弱分类器就可以使训练集和测试集的准确率都达到80%

现在我们来看一下，不同弱分类器数目情况下的AdabBoost算法的预测准确率

```
Cycles=[1,10,50,100,500,1000,10000]
train_acc=[]
test_acc=[]
for maxC in Cycles:
    a,b=calAcc(maxC)
    train_acc.append(round(a*100,2))
    test_acc.append(round(b*100,2))
df=pd.DataFrame({'分类器数目':Cycles,
                 '训练集准确率':train_acc,
                 '测试集准确率':test_acc})
df
```

	分类器数目	训练集准确率	测试集准确率
0	1	71.57	73.13
1	10	76.59	76.12
2	50	80.94	79.10
3	100	80.94	77.61
4	500	83.95	74.63
5	1000	85.95	73.13
6	10000	89.63	67.16

从上述结果中可以看出，当弱分类器数目达到50个的时候，训练集和测试集的预测准确率均达到了一个比较高的值，但是如果继续增加弱分类器数量的话，测试集的准确率反而开始下降了，这就是所谓的**过拟合**（overfitting）。

七、过拟合和欠拟合

过拟合（overfitting）和欠拟合（underfitting）是建模过程中常见的两种现象。

	欠拟合	过拟合
定义	欠拟合就是模型没有很好地捕捉到数据特征，不能够很好地拟合数据	模型过多地捕捉了训练数据中的噪音，使得模型的泛化能力变弱
训练集上表现	不好	很好
测试集上表现	不好	不好
出现原因	1. 数据集中没有或者**缺少有效特征**，也就是能让模型有效学习规律的特征 2. **模型复杂度不够**	1. 有可能是**数据不纯** 2. 数据量太少而**模型太复杂**
解决办法	1.**增加数据的特征维度**，添加一些新的特征，为模型提供更多特征，可以帮助模型更好地学习规律	1.**重新清洗数据**：模型出现过拟合有可能是原数据集不纯，噪音太多，所以可以通过重新清洗数据来消除一部分噪音
	2. **模型复杂化。**模型复杂化有两种处理手段，第一种是将同一种算法复杂化，可以采用的方式有：线性模型可以增加高次项，树模型可以增加树的深度、树的个数；第二种是弃用原来的算法，换用更高级的算法	2.**增大训练的数据量**：扩大数据量有时候并不是一个容易的事情，需要去花费时间和精力去收集和处理数据；我们可以利用现有的数据进行扩充
	3. **降低正则化约束**，正则化的目的是用来防止过拟合的，但是现在模型出现了欠拟合，则需要减少正则化参数。	3. **采用正则化**：在损失函数后面添加正则化项，可以避免训练出来的参数过大从而使模型过拟合
		4.**降低模型的复杂度**
总结	影响模型效果的因素有很多，可以尝试调整各种超参数和优化方法来看看哪些参数效果更好，但这些对于模型效果的提升可能是比较有限的。最有效提升模型效果的方法是从数据入手，其次是模型，再次是超参。一句话，最重要的是数据！数据！数据！	

八、分类器的衡量指标

前面讲的6期算法，我一直在用准确率来评价一个模型的好坏。实际上，只用准确率来评价模型的优劣是不全面的，因为准确率只涉及到所有样本中被正确分类的样本比例，而没有涵盖到样本被分错的情况。

有时候我们往往在意的就是样本被分错的概率。以病马数据集为例，如果我们预测这匹病马会死，那么他们就会对病马实施安乐死，而不是通过药物治疗来延缓其不可避免的死亡过程。但是也许我们的预测是错误的，这匹病马本来可以继续活着，毕竟我们预测的准确率只有80%。如果我们预测错误了，那么我们将会错杀一个宝贵的生命。这样的状况肯定是我们不愿意看到的。

再比如，垃圾邮件的过滤，人们对收件箱中偶尔出现几个垃圾邮件的容忍度高呢？还是正常的邮件被错分到垃圾邮件里面容忍度高呢？显然是后者最不可容忍，因为你有可能因为一封offer被错分到垃圾邮件而错过一份工作，也有可能因为一封告白邮件被错分到垃圾邮件而错过了你的Mr. Right……所以这种状况就是我们情愿被错分，也不愿意漏判，也就是说不同类别的分类代价并不相等。

因此就有**混淆矩阵**（confusion matrix）的横空出世，它可以帮我们更好的了解被错分的情况~

1. 混淆矩阵

混淆矩阵通常也被称之为"联列表" 或 "误差矩阵"，我们来看一个二分类问题的混淆矩阵：

		预测结果	
		+1	-1
实际结果	+1	f++(TP)	f+-(FN)
	-1	f-+(FP)	f--(TN)

在描述混淆矩阵，我们经常使用下面的术语。

- 真正（true positive，TP）或 f_{++} ，正样本被预测为正的个数
- 假负（false negative，FN）或 f_{+-} ，正样本被预测为负的个数
- 假正（false positive，FP）或 f_{-+} ，负样本被预测为正的个数
- 真负（true negative，TN）或 f_{--} ，负样本被预测为负的个数

准确率：所有预测正确的样本数除以总样本数

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

精确度：实际为+1且预测为+1的样本占所有预测为+1的比重

$$Precision = \frac{TP}{TP+FP}$$

召回率：也被称为敏感度，实际为+1且预测为+1的样本占所有实际为+1的比重

$$Recall = Sensitivity = \frac{TP}{TP+FN}$$

特异度：实际为-1且预测也为-1的样本占所有实际为-1的比重

$$Specificity = \frac{TN}{TN+FP}$$

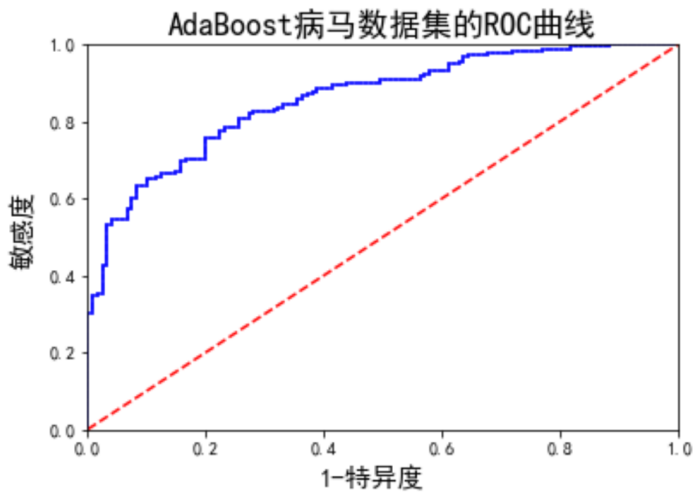
F指标：精确度和召回率的调和均值

$$F-measure = \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} = \frac{2*Precision*Recall}{Precision+Recall}$$

我们可以很容易构造一个高准确率或者高召回率的分类器，但是很难同时保证两者成立。如果将任何样本都判为正例，那么召回率可以达到百分之百，但是此时的准确率就会很低。所以构建一个同时使准确率和召回率很大的分类器是有挑战性的。

2. ROC曲线

另一个用于度量分类中的非均衡性的工具是**接受者操作特征**（receiver operating characteristic）曲线，简称ROC曲线。下图就是AdaBoost病马数据集的ROC曲线，后面我们有代码实现



我们可以看到，ROC曲线的横坐标是 【1-特异度】，纵坐标是【敏感度】也就是召回率。图中还有两条线，一条虚线一条实线。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敏感度很高的情况下也有很高的特异度，大家再看一下敏感度和特异度的定义，也就是说，我们在把正样本更多地正确分类的同时也能把负样本更多的正确分类。

下面我们来构建ROC曲线的绘制函数

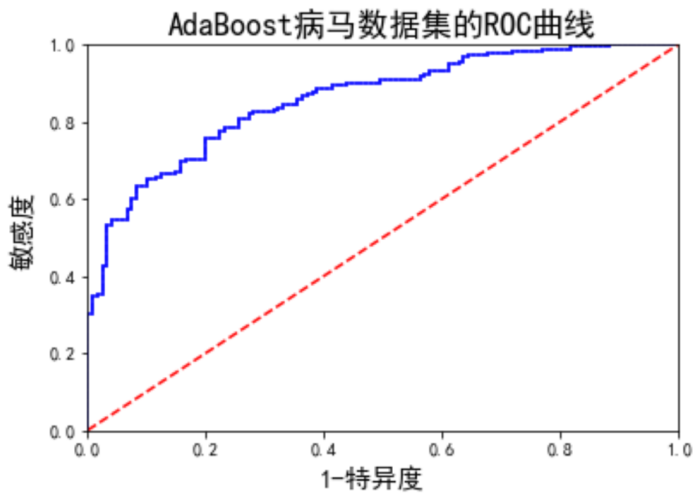
```
"""
函数功能：绘制ROC曲线
参数说明：
    xMat：特征矩阵
    yMat：标签矩阵
    maxC：最大迭代次数（即弱分类器个数）
"""
def plotROC(xMat,yMat,maxC):
    cur=(1.0,1.0)                                       #初始化x,y起点
    ySum = 0                                            #初始化累积高度
    weakClass, aggClass =Ada_train(xMat, yMat, maxC)
    P = sum(yMat==1)                                    #正样本个数
    yStep = 1 / float(P)                                #y轴步长
    xStep = 1 / float(len(yMat) - P)                    #x轴步长
    index = aggClass.T.argsort().tolist()[0]            #预测强度排序,返回的是索引
    #绘制ROC曲线
    plt.figure()
    ax = plt.subplot(111)
    for i in index:
```

```
        if yMat[i] == 1.0:
            delX = 0
            delY = yStep
        else:
            delX = xStep
            delY = 0
            ySum += cur[1]                                                    #高度累加
        ax.plot([cur[0], cur[0] - delX], [cur[1], cur[1] - delY], c = 'b') #绘制ROC
        cur = (cur[0] - delX, cur[1] - delY)                                  #更新绘制光标的位置
    ax.plot([0,1], [0,1], 'r--')
    plt.title('AdaBoost病马数据集的ROC曲线',size=18)
    plt.xlabel('1-特异度',size=15)
    plt.ylabel('敏感度',size=15)
    ax.axis([0, 1, 0, 1])
    plt.show()
```

运行函数，构建10个弱分类器，查看结果：

```
plotROC(xMat,yMat,maxC=10)
```

AUC面积为:0.86，图形如下：



3. AUC面积

对不同的ROC曲线进行比较的一个指标就是曲线下的面积（area under the curve）,简称**AUC**。一个完美分类器的AUC为1.0，随机猜测的AUC为0.5，所以我们比较两个模型之间的优劣通常使用虚线和实线之间的面积，面积越大则模型效果越好。

构建AUC的计算函数，该函数是在ROC绘制函数上稍微做了修改

```
"""
函数功能：计算ROC曲线下面积AUC
"""
def calAUC(xMat,yMat,maxC):
    cur=(1.0,1.0)
```

```
    ySum = 0
    weakClass, aggClass =Ada_train(xMat, yMat, maxC)
    P = sum(yMat==1)                                    #正样本个数
    yStep = 1 / float(P)                                #y轴步长
    xStep = 1 / float(len(yMat) - P)                    #x轴步长
    index = aggClass.T.argsort().tolist()[0]            #预测强度排序,返回的是索引
    for i in index:
        if yMat[i] == 1.0:
            delX = 0
            delY = yStep
        else:
            delX = xStep
            delY = 0
            ySum += cur[1]                              #高度累加
        cur = (cur[0] - delX, cur[1] - delY)            #更新绘制光标的位置
    AUC= ySum * xStep
    print(f'AUC面积为:{round(AUC,2)}')
    return AUC
```

运行函数，构建10个弱分类器，查看结果：

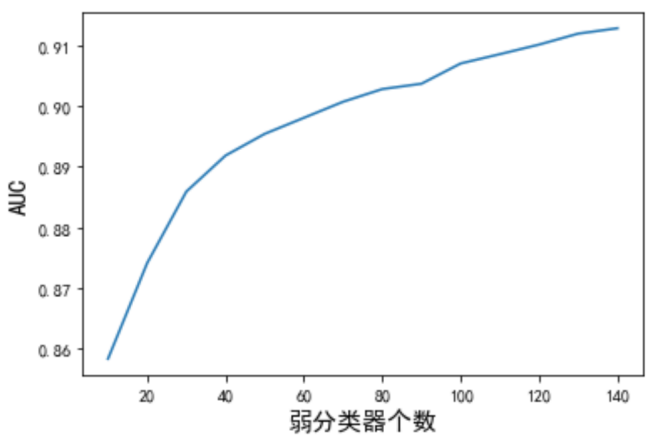
```
calAUC(xMat,yMat,10)
```

构建10个弱分类器，得到AUC的结果为0.86，增加弱分类器的数量，AUC的结果会不会变得更好呢？

我们来尝试一下：

```
AUC=[]
for i in np.arange(10,150,10):
    auc= calAUC(xMat,yMat,i)
    AUC.append(auc)
plt.plot(np.arange(10,150,10),AUC,c='b')
plt.xlabel('弱分类器个数',size=15)
plt.ylabel('AUC',size=15)
plt.show()
```

结果如下：



从图中可以看到，随着弱分类器个数的增加，AUC的值也在逐渐的增加。那是不是说明，弱分类器个数越多越好呢？答案肯定不是，因为前面我们在讲过拟合的时候有提到，弱分类器的数量增加到一定程度之后就会出现过拟合。那究竟弱分类器的个数应该定为多少合适？AUC的曲线图可以告诉你答案。大家再仔细观察这张图，曲线的斜率代表增加的速度对吧，可以发现在弱分类器个数达到40之后，AUC曲线的斜率变小了，也就是说AUC增加的速度变慢了。那么这个AUC改变的这个拐点，就是我们弱分类器的最佳取值。

九、样本不均衡问题

机器学习中常常会会假设我们的训练样本各类别数目是均衡的，但是实际场景中常常无法满足这个假设。例如，在传统制造业，不合格产品的数量远远低于合格产品的数量（不满足这个条件的企业早就倒闭了~~~）；在信用卡欺诈检测中，合法交易远远多于欺诈交易。在这两个例子中，不平衡的程度随应用不同而不同——一个达到六西格玛标准的制造业，每一百万件产品中仅有4个不合格品；而信用卡欺诈的量级可能是万分之一（不同银行有不同的业务基准）。尽管它们不常出现，但是确是我们重点关注的对象，也就是说，这种稀有类的正确分类比多数类的正常分类更有价值。但是由于样本不平衡的问题，模型常常会聚焦于样本多的那一类。所以我们就需要对样本不均衡这样的问题进行处理。

1. 样本层面的处理

数据层面处理方法多借助数据采样法使整体训练集样本趋于平衡，即各类样本数基本一致。常用的方法有**欠采样**（undersampling）或者 **过采样**（oversampling）。

欠采样是通过减少丰富类的大小来平衡数据集，当数据量足够时使用此方法。通过保存所有稀有类样本，并在丰富类别中随机选择与稀有类别样本相等数量的样本，可以检索平衡的新数据集以进一步建模。

过采样是通过增加稀有样本的数量来平衡数据集，当数据量不足时使用此方法。比如可以将稀有样本进行复制来增加稀有样本的数量。

欠采样和过采样这两种方法相比而言，都没有绝对的优势，所以有时候也会将两种采样方法结合起来使用。

此外还有一种方法，使用**SMOTE算法**来构造样本。SMOTE全称是Synthetic Minority Oversampling Technique即合成少数类过采样技术，它是基于随机过采样算法的一种改进方案，由于随机过采样采取简单复制样本的策略来增加少数类样本，这样容易产生模型过拟合的问题，即使得模型学习到的信息过于特别(Specific)而不够泛化(General)。SMOTE算法的基本思想：

- 基于距离度量的方式计算两个或多个稀有类样本之间的相似性。
- 然后选择其中的一个样本作为基础样本。
- 再在邻居样本中随机选取一定数量的样本对那个基础样本的一个属性进行噪声。
- 每次处理一个属性，通过这样的方式产生新生数据。

2. 算法层面的处理

对于样本不平衡问题，除了在采样上做处理，还可以在算法上进行优化。

方法一：**模型融合** （bagging的思想 ）

思路：从丰富类样本中随机的选取（有放回的选取）和稀有类等量样本的数据。和稀有类样本组合成新的训练集。这样我们就产生了多个训练集，并且是互相独立的，然后训练得到多个分类器。

若是分类问题，就把多个分类器投票的结果（少数服从多数）作为分类结果。

若是回归问题，就将均值作为最后结果。

方法二：**增量模型** （boosting的思想）

思路：使用全部的样本作为训练集，得到分类器L1，从L1正确分类的样本中和错误分类的样本中各抽取50%的数据，即循环的一边采样一个。此时训练样本是平衡的。训练得到的分类器作为L2，从L1和L2分类结果中，选取结果不一致的样本作为训练集得到分类器L3，最后投票L1,L2,L3结果得到最后的分类结果。

3. 改变样本的权重

有一种称为代价敏感的学习（cost-sensitive learning）可以用于处理非均衡分类的问题。

下图是两个代价矩阵

代价矩阵1		预测结果	
		+1	-1
实际结果	+1	0	1
	-1	1	0

代价矩阵2		预测结果	
		+1	-1
实际结果	+1	-5	1
	-1	50	0

代价矩阵1是我们到目前为止分类器的代价矩阵（代价不是0就是1），我们可以基于该代价矩阵计算其总代价：$TP*0+FN*1+FP*1+TN*0$。

我们也可以通过人为的改变代价矩阵，来使得我们分类错误的代价变得不同。基于代价矩阵2的总代价为：$TP*(-5)+FN*1+FP*50+TN*0$，这样其实我们人为增大了FP（负样本被预测为正）的代价。

如果在构建分类器时，知道了这些代价值，那么就可以选择付出最小代价的分类器。

在分类算法中，我们有很多方法可以引入代价信息。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。在SVM中，可以在代价函数中对于不同的类别选择不同的参数C。上述做法就会给较小类更多的权重，即在训练时，小类当中只允许更少的错误。

十、实例：泰坦尼克幸存者预测

泰坦尼克号的沉没是世界上最严重的海难事故之一，今天我们通过adaboost算法来预测一下哪些人可能成为幸存者。 数据集来自https://www.kaggle.com/c/titanic，其中包含两个csv格式文件，data为我们接下来要使用的数据，test为kaggle提供的测试集。

特征名	描述	类型
survival	Survival (**标签**)	number
	Values: (0 = No; 1 = Yes)	
pclass	Passenger Class	number
	Values: (1 = 1st; 2 = 2nd; 3 = 3rd)	
name	Name	string
sex	Sex	string
	Values: (female, male)	
age	Age	number
sibsp	Number of Siblings/Spouses Aboard	number
parch	Number of Parents/Children Aboard	number
ticket	Ticket Number	string
fare	Passenger Fare	number
cabin	Cabin	string
embarked	Port of Embarkation	character
	Values: (C = Cherbourg; Q = Queenstown; S = Southampton)	

1. 导入数据集并进行预处理

```
#导入数据集
import pandas as pd
data = pd.read_csv('Taitanic_data/data.csv')

#探索数据
data.head()
data.shape
data.info()
data.isnull().sum() #查看缺失值

#数据预处理
#删除缺失值多，目测对建模没用的列
data.drop(['PassengerId','Name','Cabin','Ticket'],inplace=True,axis=1)
#处理缺失值
data["Age"] = data["Age"].fillna(data["Age"].mean())
data = data.dropna()
#将二分类变量转换为数值型变量
data['Sex']=(data['Sex']=='male').astype('int')
#将三分类变量转换为数值型变量
labels = data['Embarked'].unique().tolist()
```

```
data['Embarked'] = data['Embarked'].apply(lambda x: labels.index(x))
#查看处理后的数据集
data.head()
data.shape
```

2. 划分训练集和测试集

```
#提取特征矩阵和标签矩阵
X = data.iloc[:,data.columns != "Survived"]
y = data.iloc[:,data.columns == "Survived"]

#划分训练集和测试集
from sklearn.model_selection import train_test_split
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X,y,test_size=0.25)

#修正测试集和训练集的索引
for i in [Xtrain, Xtest, Ytrain, Ytest]:
    i.index = range(i.shape[0])

#查看分好的训练集和测试集
Xtrain.head()
```

3. 建立AdaBoost分类模型

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
AdaBoostClassifier
import warnings
warnings.filterwarnings('ignore')

#带入模型粗跑一下
ada = AdaBoostClassifier(random_state=25)
ada = ada.fit(Xtrain, Ytrain)
score_ada = ada.score(Xtest, Ytest)
score_ada

#交叉验证
from sklearn.model_selection import cross_val_score
scores = cross_val_score(ada, Xtrain, Ytrain, cv=5)
print(f'这个模型的准确率为{round(scores.mean() * 100,2)}%(+/- {round(scores.std()*2
*100,2)}%)')
```

4. 其他集成算法结果

```
#随机森林
rfc = RandomForestClassifier(random_state=25)
rfc = rfc.fit(Xtrain, Ytrain)
score_rfc = rfc.score(Xtest, Ytest)
score_rfc

#梯度提升
gbc = GradientBoostingClassifier(random_state=25)
gbc = gbc.fit(Xtrain, Ytrain)
score_gbc = gbc.score(Xtest, Ytest)
score_gbc

#AdaBoost与随机森林结合
model = AdaBoostClassifier(RandomForestClassifier(n_estimators = 1000),
                           algorithm="SAMME",
                           n_estimators=500)
model = model.fit(Xtrain, Ytrain)
score_AdaRfc = model.score(Xtest, Ytest)
score_AdaRfc
```

其他

- 菊安酱的直播间：https://live.bilibili.com/14988341
- 下周一（2018/12/17）将讲解**线性回归**，欢迎各位进入菊安酱的直播间观看直播
- 如有问题，可以给我留言哦~

【参考资料】

[1] bagging和boosting算法（集成学习算法）：

https://blog.csdn.net/chenyukuai6625/article/details/73692347

[2] 集成学习原理小结

https://www.cnblogs.com/pinard/p/6131423.html

[3] 不平衡样本的处理

https://blog.csdn.net/siyue0211/article/details/80318999

[4] 机器学习中样本不平衡问题

https://www.jianshu.com/p/71eea3555dbf

[5] 如何解决样本不均衡问题

https://www.cnblogs.com/lyr2015/p/8711120.html