《机器学习实战》学习笔记（七）：利用AdaBoost 元算法提高分类性能

原创 我是管小亮 2019-09-02 16:38:34 2232 收藏 11 版权

分类专栏： Machine Learning♡ 文章标签： 机器学习 机器学习实战 AdaBoost 利用AdaBoost 元算法提高分类性能 学习笔记

欢迎关注WX公众号：【程序员管小亮】

【机器学习】《机器学习实战》读书笔记及代码 总目录

- https://blog.csdn.net/TeFuirnever/article/details/99701256

GitHub代码地址：

- https://github.com/TeFuirnever/Machine-Learning-in-Action

- 《机器学习》周志华西瓜书学习笔记（八）：集成学习
- 《机器学习》周志华西瓜书习题参考答案：第8章 - 集成学习

目录

本章内容

- 组合相似的分类器来提高分类性能
- 应用AdaBoost算法
- 处理非均衡分类问题

1、基于数据集多重抽样的分类器 - 集成方法

当做重要决定时，大家可能都会考虑吸取多个专家而不只是一个人的意见。机器学习处理问题时又何尝不是如此？我们可以很自然地将不同的分类器组合起来，而这种组合结果则被称为 **集成方法(ensemble method)** 或者 **元算法(meta-algorithm)**。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（ensemble method） 通过组合多个学习器来完成学习任务，有点“三个臭皮匠顶个诸葛亮”的意味。基分类器一般采用的是 **弱可学习（weakly learnable）** 分类器，通过 **集成方法**，组合成一个 **强可学习（strongly learnable）** 分类器。**弱可学习** 是指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** 指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，这是因为大部分基分类器都分类错误的概率远低于单一基分类器的。

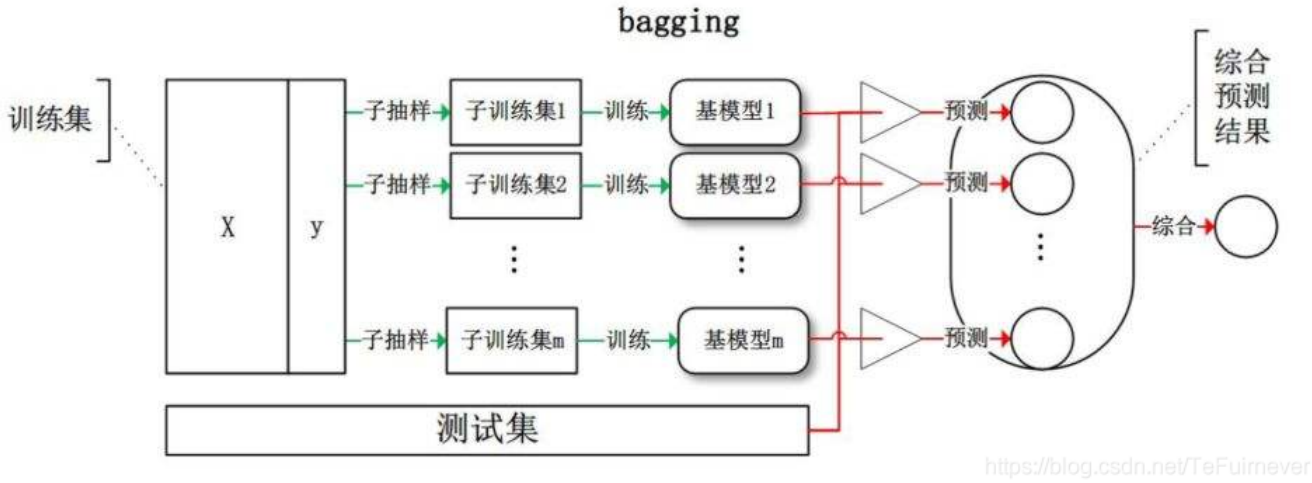
集成方法 主要包括 **Bagging** 和 **Boosting** 两种方法，**Bagging** 和 **Boosting** 都是将已有的分类或回归算法通过一定方式组合起来，形成一个性能更加强大的分类器，更准确的说这是一种分类算法的组装方法，即将 **弱分类器** 组装成 **强分类器** 的方法。

1）bagging：基于数据随机重抽样的分类器构建方法

自举汇聚法（bootstrap aggregating），也称为bagging方法，是在从原始数据集选择S次后得到S个新数据集的一种技术。新数据集和原数据集的大小相等。每个数据集都是通过在原始数据集中随机选择一个样本来进行替换而得到的。这里的替换就意味着可以多次地选择同一样本。这一性质就允许新数据集中可以有重复的值，而原始数据集的某些值在新集合中则不再出现。在S个数据集建好之后，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。当要对新数据进行分类时，就可以应用这S个分类器进行分类。与此同时，**选择分类器投票结果中最多的类别作为最后的分类结果。**

当然，还有一些更先进的bagging方法，比如 **随机森林（random forest）**。接下来我们将注意力转向一个与bagging类似的集成分类器方法boosting。

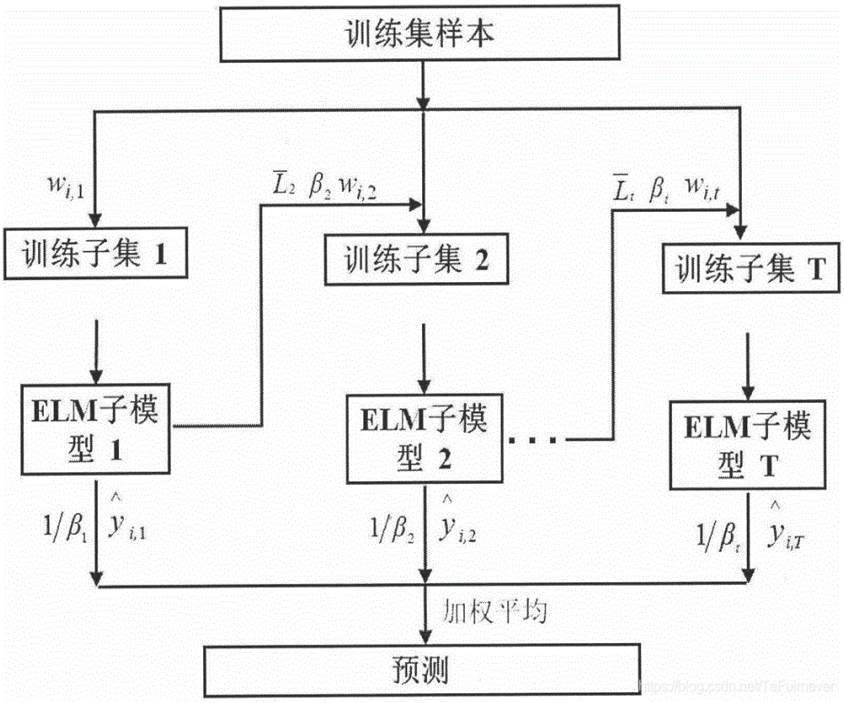
> Bagging
> 1.通过降低基分类器的方差，改善了泛化误差。
> 2.性能依赖于基分类器的稳定性，如果基分类器不稳定，bagging有助于降低训练数据的随机波动导致的误差；如果稳定，则集成分类器的误差主要由基分类器的偏倚引起。
> 3.由于每个样本被选中的概率相同，因此bagging并不侧重于训练数据集中的任何特定实例。



2）boosting

boosting是一种与bagging很类似的技术。不论是在boosting还是bagging当中，所使用的多个分类器的类型都是一致的。但是在前者当中，不同的分类器是通过串行训练而获得的，每个新分类器都根据已训练出的分类器的性能来进行训练。

boosting是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。由于boosting分类的结果是基于所有分类器的加权求和结果的，因此boosting与bagging不太一样。**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轮迭代中的成功度。**



3）二者的区别

- 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。
- 样例权重：
  - 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。
- 预测函数：
  - 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。
- 并行计算：
  - 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。

2、提升分类器性能利器 - AdaBoost

boosting方法拥有多个版本，本章将只关注其中一个最流行的版本AdaBoost。

> 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。
> 缺点：对离群点敏感。
> 适用数据类型：数值型和标称型数据。

> AdaBoost的一般流程
> (1) 收集数据：可以使用任意方法。
> (2) 准备数据：依赖于所使用的弱分类器类型，本章使用的是单层决策树，这种分类器可以处理任何数据类型。当然也可以使用任意分类器作为弱分类器，第2章到第6章中的任一分类器都可以充当弱分类器。作为弱分类器，简单分类器的效果更好。
> (3) 分析数据：可以使用任意方法。
> (4) 训练算法：AdaBoost的大部分时间都用在训练上，分类器将多次在同一数据集上训练弱分类器。
> (5) 测试算法：计算分类的错误率。
> (6) 使用算法：同SVM一样，AdaBoost预测两个类别中的一个。如果想把它应用到多个类别的场合，那么就要像多类SVM中的做法一样对AdaBoost进行修改。

能否使用弱分类器和多个实例来构建一个强分类器？这是一个非常有趣的理论问题。这里的“弱”意味着分类器的性能比随机猜测要略好，但是也不会好太多。这就是说，在二分类情况下弱分类器的错误率会高于50%，而“强”分类器的错误率将会低很多。AdaBoost算法即脱胎于上述理论问题。

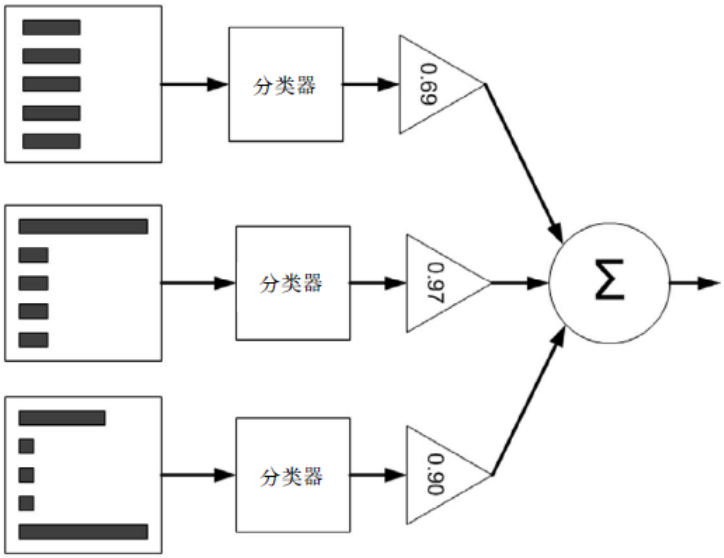
AdaBoost 是 **adaptive boosting（自适应boosting）** 的缩写，其运行过程如下：训练数据中的每个样本，并赋予其一个权重，这些权重构成了向量D。一开始，这些权重都初始化成相等值。首先在训练数据上训练出一个弱分类器并计算该分类器的错误率，然后在同一数据集上再次训练弱分类器。在分类器的第二次训练当中，将会重新调整每个样本的权重，其中 **第一次分对的样本的权重将会降低，而第一次分错的样本的权重将会提高。**为了从所有弱分类器中得到最终的分类结果，AdaBoost为每个分类器都分配了一个权重值alpha，这些alpha值是基于每个弱分类器的错误率进行计算的。其中，错误率ε的定义为：

$$\varepsilon = \frac{\text{未正确分类的样本数目}}{\text{所有样本数目}}$$

而alpha的计算公式如下：

$$\alpha = \frac{1}{2}\ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$$

AdaBoost算法的流程如图所示。



AdaBoost算法的示意图。左边是数据集，其中直方图的不同宽度表示每个样例上的不同权重。在经过一个分类器之后，加权的预测结果会通过三角形中的alpha值进行加权。每个三角形中输出的加权结果在圆形中求和，从而得到最终的输出结果

计算出alpha值之后，可以对权重向量D进行更新，以使得那些正确分类的样本的权重降低而错分样本的权重升高。D的计算方法如下。

如果某个样本被正确分类，那么该样本的权重更改为：

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

而如果某个样本被错分，那么该样本的权重更改为：

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

在计算出D之后，AdaBoost又开始进入下一轮迭代。AdaBoost算法会不断地重复训练和调整权重的过程，直到训练错误率为0或者弱分类器的数目达到用户的指定值为止。

AdaBoost算法总结如下：

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
- Update:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Figure 1: The boosting algorithm AdaBoost.

3、基于单层决策树构建弱分类器

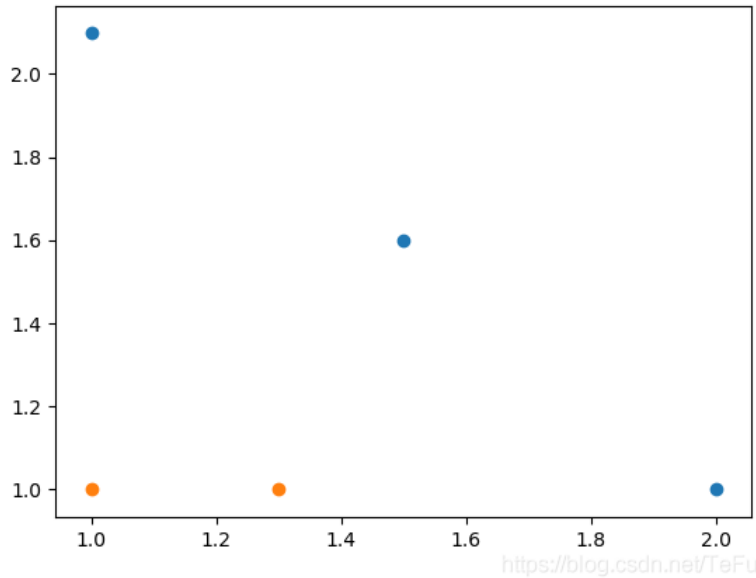
建立AdaBoost 算法之前，必须先建立弱分类器，并保存样本的权重。弱分类器使用 **单层决策树（decision stump）**，也称 **决策树桩**，它是一种简单的决策树，通过给定的阈值，进行分类。构建一个单层决策树，而它仅基于单个特征来做决策。由于这棵树只有一次分裂过程，因此它实际上就是一个树桩。

1）数据集可视化

为了训练单层决策树，我们需要创建一个训练集，编写代码如下：

```
import numpy as np
import matplotlib.pyplot as plt

"""
Parameters:
    无
Returns:
    dataMat - 数据矩阵
    classLabels - 数据标签
"""
# 创建单层决策树的数据集
def loadSimpData():
    datMat = np.matrix([[ 1. ,  2.1],
        [ 1.5,  1.6],
        [ 1.3,  1. ],
        [ 1. ,  1. ],
        [ 2. ,  1. ]])
    classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]
    return datMat,classLabels

"""
Parameters:
    dataMat - 数据矩阵
    labelMat - 数据标签
Returns:
    无
"""
# 数据可视化
def showDataSet(dataMat, labelMat):
    data_plus = []                                  #正样本
    data_minus = []                                 #负样本
    for i in range(len(dataMat)):
        if labelMat[i] > 0:
```

```
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
    data_plus_np = np.array(data_plus)                                              #转换为numpy矩阵
    data_minus_np = np.array(data_minus)                                            #转换为numpy矩阵
    plt.scatter(np.transpose(data_plus_np)[0], np.transpose(data_plus_np)[1])    #正样本散点图
    plt.scatter(np.transpose(data_minus_np)[0], np.transpose(data_minus_np)[1]) #负样本散点图
    plt.show()


if __name__ == '__main__':
    dataArr,classLabels = loadSimpData()
    showDataSet(dataArr,classLabels)
```



可以发现，如果想要试着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一个著名问题。通过使用多棵单层决策树，我们就可以构建出一个能够对该数据集完全正确分类的分类器。

2）构建单层决策树

有了数据，接下来就可以通过构建多个函数来建立单层决策树。第一个函数将用于测试是否有某个值小于或者大于我们正在测试的阈值。第二个函数则更加复杂一些，它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。

```
import numpy as np
import matplotlib.pyplot as plt

"""
Parameters:
    无
Returns:
    dataMat - 数据矩阵
    classLabels - 数据标签
"""
# 创建单层决策树的数据集
def loadSimpData():
    datMat = np.matrix([[ 1. ,  2.1],
        [ 1.5,  1.6],
        [ 1.3,  1. ],
        [ 1. ,  1. ],
        [ 2. ,  1. ]])
    classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]
    return datMat,classLabels

"""
Parameters:
```

```
    dataMatrix - 数据矩阵
    dimen - 第dimen列，也就是第几个特征
    threshVal - 阈值
    threshIneq - 标志
Returns:
    retArray - 分类结果
"""
# 单层决策树分类函数
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = np.ones((np.shape(dataMatrix)[0],1))         #初始化retArray为1
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0   #如果小于阈值,则赋值为-1
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0    #如果大于阈值,则赋值为-1
    return retArray

"""
Parameters:
    dataArr - 数据矩阵
    classLabels - 数据标签
    D - 样本权重
Returns:
    bestStump - 最佳单层决策树信息
    minError - 最小误差
    bestClasEst - 最佳的分类结果
"""
# 找到数据集上最佳的单层决策树
def buildStump(dataArr,classLabels,D):
    dataMatrix = np.mat(dataArr); labelMat = np.mat(classLabels).T
    m,n = np.shape(dataMatrix)
    numSteps = 10.0; bestStump = {}; bestClasEst = np.mat(np.zeros((m,1)))
    minError = float('inf')                                                 #最小误差初始化为正无穷大
    for i in range(n):                                                      #遍历所有特征
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max()      #找到特征中最小的值和最大值
        stepSize = (rangeMax - rangeMin) / numSteps                         #计算步长
        for j in range(-1, int(numSteps) + 1):
            for inequal in ['lt', 'gt']:                                    #大于和小于的情况，均遍历。lt:less than, gt:
                threshVal = (rangeMin + float(j) * stepSize)                #计算阈值
                predictedVals = stumpClassify(dataMatrix, i, threshVal, inequal)#计算分类结果
                errArr = np.mat(np.ones((m,1)))                             #初始化误差矩阵
                errArr[predictedVals == labelMat] = 0                       #分类正确的,赋值为0
                weightedError = D.T * errArr                                #计算误差
                print("split: dim %d, thresh %.2f, thresh ineqal: %s, the weighted error is %.3f" % (i, threshVal, inequa
                if weightedError < minError:                                #找到误差最小的分类方式
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump,minError,bestClasEst


if __name__ == '__main__':
    dataArr,classLabels = loadSimpData()
    D = np.mat(np.ones((5, 1)) / 5)
    bestStump,minError,bestClasEst = buildStump(dataArr,classLabels,D)
    print('bestStump:\n', bestStump)
    print('minError:\n', minError)
    print('bestClasEst:\n', bestClasEst)
```

```
split: dim 1, thresh 1.77, thresh ineqal: gt, the weighted error is 0.600
split: dim 1, thresh 1.88, thresh ineqal: lt, the weighted error is 0.400
split: dim 1, thresh 1.88, thresh ineqal: gt, the weighted error is 0.600
split: dim 1, thresh 1.99, thresh ineqal: lt, the weighted error is 0.400
split: dim 1, thresh 1.99, thresh ineqal: gt, the weighted error is 0.600
split: dim 1, thresh 2.10, thresh ineqal: lt, the weighted error is 0.600
split: dim 1, thresh 2.10, thresh ineqal: gt, the weighted error is 0.400
bestStump:
 {'dim': 0, 'thresh': 1.3, 'ineq': 'lt'}
minError:
 [[ 0.2]]
bestClasEst:
 [[-1.]
 [ 1.]
 [-1.]
 [-1.]
 [ 1.]]
```


上述单层决策树的生成函数是决策树的一个简化版本。它就是所谓的弱学习器，即弱分类算法。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。

4、完整AdaBoost 算法的实现

整个实现的伪代码如下：

```
对每次迭代：
        利用buildStump()函数找到最佳的单层决策树
        将最佳单层决策树加入到单层决策树数组
        计算alpha
        计算新的权重向量D
        更新累计类别估计值
        如果错误率等于0.0，则退出循环
```

使用AdaBoost算法提升分类器性能，编写代码如下：

```python
import numpy as np
import matplotlib.pyplot as plt

"""
Parameters:
    无
Returns:
    dataMat - 数据矩阵
    classLabels - 数据标签
"""
# 创建单层决策树的数据集
def loadSimpData():
    datMat = np.matrix([[ 1. ,  2.1],
        [ 1.5,  1.6],
        [ 1.3,  1. ],
        [ 1. ,  1. ],
        [ 2. ,  1. ]])
    classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]
    return datMat,classLabels

"""
Parameters:
    dataMatrix - 数据矩阵
    dimen - 第dimen列，也就是第几个特征
    threshVal - 阈值
    threshIneq - 标志
Returns:
    retArray - 分类结果
"""
# 单层决策树分类函数
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = np.ones((np.shape(dataMatrix)[0],1))        #初始化retArray为1
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0 #如果小于阈值,则赋值为-1
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0  #如果大于阈值,则赋值为-1
```

```
    return retArray

"""
Parameters:
    dataArr - 数据矩阵
    classLabels - 数据标签
    D - 样本权重
Returns:
    bestStump - 最佳单层决策树信息
    minError - 最小误差
    bestClasEst - 最佳的分类结果
"""
# 找到数据集上最佳的单层决策树
def buildStump(dataArr,classLabels,D):
    dataMatrix = np.mat(dataArr); labelMat = np.mat(classLabels).T
    m,n = np.shape(dataMatrix)
    numSteps = 10.0; bestStump = {}; bestClasEst = np.mat(np.zeros((m,1)))
    minError = float('inf')                                                 #最小误差初始化为正无穷大
    for i in range(n):                                                      #遍历所有特征
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max()  #找到特征中最小的值和最大值
        stepSize = (rangeMax - rangeMin) / numSteps                         #计算步长
        for j in range(-1, int(numSteps) + 1):
            for inequal in ['lt', 'gt']:                                    #大于和小于的情况，均遍历。lt:less than, gt
                threshVal = (rangeMin + float(j) * stepSize)                #计算阈值
                predictedVals = stumpClassify(dataMatrix, i, threshVal, inequal) #计算分类结果
                errArr = np.mat(np.ones((m,1)))                             #初始化误差矩阵
                errArr[predictedVals == labelMat] = 0                       #分类正确的,赋值为0
                weightedError = D.T * errArr                                #计算误差
                print("split: dim %d, thresh %.2f, thresh ineqal: %s, the weighted error is %.3f" % (i, threshVal, inequa
                if weightedError < minError:                                #找到误差最小的分类方式
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump, minError, bestClasEst

def adaBoostTrainDS(dataArr, classLabels, numIt = 40):
    weakClassArr = []
    m = np.shape(dataArr)[0]
    D = np.mat(np.ones((m, 1)) / m)                                         #初始化权重
    aggClassEst = np.mat(np.zeros((m,1)))
    for i in range(numIt):
        bestStump, error, classEst = buildStump(dataArr, classLabels, D)  #构建单层决策树
        print("D:",D.T)
        alpha = float(0.5 * np.log((1.0 - error) / max(error, 1e-16)))    #计算弱学习算法权重alpha,使error不等于0,因为分母不
        bestStump['alpha'] = alpha                                          #存储弱学习算法权重
        weakClassArr.append(bestStump)                                      #存储单层决策树
        print("classEst: ", classEst.T)
        expon = np.multiply(-1 * alpha * np.mat(classLabels).T, classEst) #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
        #计算AdaBoost误差，当误差为0的时候，退出循环
        aggClassEst += alpha * classEst
        print("aggClassEst: ", aggClassEst.T)
        aggErrors = np.multiply(np.sign(aggClassEst) != np.mat(classLabels).T, np.ones((m,1)))#计算误差
        errorRate = aggErrors.sum() / m
        print("total error: ", errorRate)
        if errorRate == 0.0: break                                          #误差为0，退出循环
    return weakClassArr, aggClassEst


if __name__ == '__main__':
    dataArr,classLabels = loadSimpData()
    weakClassArr, aggClassEst = adaBoostTrainDS(dataArr, classLabels)
    print(weakClassArr)
    print(aggClassEst)
```

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。（正确：+ + - - +）

```
split: dim 1, thresh 2.10, thresh ineqal: lt, the weighted error is 0.600
split: dim 1, thresh 2.10, thresh ineqal: gt, the weighted error is 0.400
D: [[ 0.2  0.2  0.2  0.2  0.2]]                          第一轮迭代
classEst:  [[-1.  1. -1. -1.  1.]]
aggClassEst:  [[-0.69314718  0.69314718 -0.69314718 -0.69314718  0.69314718]]
total error:  0.2
split: dim 0, thresh 0.90, thresh ineqal: lt, the weighted error is 0.250
split: dim 0, thresh 0.90, thresh ineqal: gt, the weighted error is 0.750
```

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，我们就会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。（正确：+ + - - +）

```
split: dim 1, thresh 2.10, thresh ineqal: lt, the weighted error is 0.750
split: dim 1, thresh 2.10, thresh ineqal: gt, the weighted error is 0.250
D: [[0.5    0.125  0.125  0.125  0.125]]                  第二轮迭代
classEst:  [[ 1.  1. -1. -1. -1.]]
aggClassEst:  [[ 0.27980789  1.66610226 -1.66610226 -1.66610226 -0.27980789]]
total error:  0.2
split: dim 0, thresh 0.90, thresh ineqal: lt, the weighted error is 0.143
split: dim 0, thresh 0.90, thresh ineqal: gt, the weighted error is 0.857
```

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。（正确：+ + - - +）

```
split: dim 1, thresh 2.10, thresh ineqal: lt, the weighted error is 0.857
split: dim 1, thresh 2.10, thresh ineqal: gt, the weighted error is 0.143
D: [[ 0.28571429  0.07142857  0.07142857  0.07142857  0.5       ]]
classEst:  [[ 1.  1.  1.  1.  1.]]
aggClassEst:  [[ 1.17568763  2.56198199 -0.77022252 -0.77022252  0.61607184]]
total error:  0.0
[{'dim': 0, 'thresh': 1.3, 'ineq': 'lt', 'alpha': 0.6931471805599453}, {'dim'
[[ 1.17568763]                                           第三轮迭代
 [ 2.56198199]
 [-0.77022252]
 [-0.77022252]
 [ 0.61607184]]                          https://blog.csdn.net/TeFuirnever
```

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。

进行测试，编写代码如下：

```python
import numpy as np
import matplotlib.pyplot as plt

"""
Parameters:
    无
Returns:
    dataMat - 数据矩阵
    classLabels - 数据标签
"""
# 创建单层决策树的数据集
def loadSimpData():
    datMat = np.matrix([[ 1. ,  2.1],
        [ 1.5,  1.6],
        [ 1.3,  1. ],
        [ 1. ,  1. ],
        [ 2. ,  1. ]])
    classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]
    return datMat,classLabels

"""
Parameters:
    dataMat - 数据矩阵
    labelMat - 数据标签
```

```
Returns:
    无
"""
# 数据可视化
def showDataSet(dataMat, labelMat):
    data_plus = []                                                                  #正样本
    data_minus = []                                                                 #负样本
    for i in range(len(dataMat)):
        if labelMat[i] > 0:
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
    data_plus_np = np.array(data_plus)                                              #转换为numpy矩阵
    data_minus_np = np.array(data_minus)                                            #转换为numpy矩阵
    plt.scatter(np.transpose(data_plus_np)[0], np.transpose(data_plus_np)[1])   #正样本散点图
    plt.scatter(np.transpose(data_minus_np)[0], np.transpose(data_minus_np)[1]) #负样本散点图
    plt.show()

"""
Parameters:
    dataMatrix - 数据矩阵
    dimen - 第dimen列，也就是第几个特征
    threshVal - 阈值
    threshIneq - 标志
Returns:
    retArray - 分类结果
"""
# 单层决策树分类函数
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = np.ones((np.shape(dataMatrix)[0],1))        #初始化retArray为1
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0  #如果小于阈值,则赋值为-1
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0   #如果大于阈值,则赋值为-1
    return retArray

"""
Parameters:
    dataArr - 数据矩阵
    classLabels - 数据标签
    D - 样本权重
Returns:
    bestStump - 最佳单层决策树信息
    minError - 最小误差
    bestClasEst - 最佳的分类结果
"""
# 找到数据集上最佳的单层决策树
def buildStump(dataArr,classLabels,D):
    dataMatrix = np.mat(dataArr); labelMat = np.mat(classLabels).T
    m,n = np.shape(dataMatrix)
    numSteps = 10.0; bestStump = {}; bestClasEst = np.mat(np.zeros((m,1)))
    minError = float('inf')                                                         #最小误差初始化为正无穷大
    for i in range(n):                                                              #遍历所有特征
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max()      #找到特征中最小的值和最大值
        stepSize = (rangeMax - rangeMin) / numSteps                                 #计算步长
        for j in range(-1, int(numSteps) + 1):
            for inequal in ['lt', 'gt']:                                            #大于和小于的情况，均遍历。lt:less than, gt:
                threshVal = (rangeMin + float(j) * stepSize)                        #计算阈值
                predictedVals = stumpClassify(dataMatrix, i, threshVal, inequal)#计算分类结果
                errArr = np.mat(np.ones((m,1)))                                     #初始化误差矩阵
                errArr[predictedVals == labelMat] = 0                               #分类正确的,赋值为0
                weightedError = D.T * errArr                                        #计算误差
                # print("split: dim %d, thresh %.2f, thresh ineqal: %s, the weighted error is %.3f" % (i, threshVal, ineq
                if weightedError < minError:                                    #找到误差最小的分类方式
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump, minError, bestClasEst

"""
```

```
Parameters:
    dataArr - 数据矩阵
    classLabels - 数据标签
    numIt - 最大迭代次数
Returns:
    weakClassArr - 训练好的分类器
    aggClassEst - 类别估计累计值
"""
# 使用AdaBoost算法提升弱分类器性能
def adaBoostTrainDS(dataArr, classLabels, numIt = 40):
    weakClassArr = []
    m = np.shape(dataArr)[0]
    D = np.mat(np.ones((m, 1)) / m)                                    #初始化权重
    aggClassEst = np.mat(np.zeros((m,1)))
    for i in range(numIt):
        bestStump, error, classEst = buildStump(dataArr, classLabels, D)  #构建单层决策树
        # print("D:",D.T)
        alpha = float(0.5 * np.log((1.0 - error) / max(error, 1e-16)))    #计算弱学习算法权重alpha,使error不等于0,因为分母不
        bestStump['alpha'] = alpha                                     #存储弱学习算法权重
        weakClassArr.append(bestStump)                                 #存储单层决策树
        # print("classEst: ", classEst.T)
        expon = np.multiply(-1 * alpha * np.mat(classLabels).T, classEst) #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
        #计算AdaBoost误差，当误差为0的时候，退出循环
        aggClassEst += alpha * classEst                                #计算类别估计累计值
        # print("aggClassEst: ", aggClassEst.T)
        aggErrors = np.multiply(np.sign(aggClassEst) != np.mat(classLabels).T, np.ones((m,1))) #计算误差
        errorRate = aggErrors.sum() / m
        # print("total error: ", errorRate)
        if errorRate == 0.0: break                                     #误差为0，退出循环
    return weakClassArr, aggClassEst

"""
Parameters:
    datToClass - 待分类样例
    classifierArr - 训练好的分类器
Returns:
    分类结果
"""
# AdaBoost分类函数
def adaClassify(datToClass,classifierArr):
    dataMatrix = np.mat(datToClass)
    m = np.shape(dataMatrix)[0]
    aggClassEst = np.mat(np.zeros((m,1)))
    for i in range(len(classifierArr)):                                    #遍历所有分类器，进行分类
        classEst = stumpClassify(dataMatrix, classifierArr[i]['dim'], classifierArr[i]['thresh'], classifierArr[i]['ineq'
        aggClassEst += classifierArr[i]['alpha'] * classEst
        print(aggClassEst)
    return np.sign(aggClassEst)


if __name__ == '__main__':
    dataArr,classLabels = loadSimpData()
    weakClassArr, aggClassEst = adaBoostTrainDS(dataArr, classLabels)
    print(adaClassify([[0,0],[5,5]], weakClassArr))
```

```
[[-0.69314718]
 [ 0.69314718]]
[[-1.66610226]
 [ 1.66610226]]
[[-2.56198199]
 [ 2.56198199]]
[[-1.]
 [ 1.]]
```

测试需要在之前代码的基础上，添加adaClassify()函数，然后遍历所有训练得到的弱分类器，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，(0,0)属于负类，所以是-1，(5,5)属于正类，所以是+1。

5、示例：在一个难数据集上应用AdaBoost

本节我们将在第4章给出的马疝病数据集上应用AdaBoost分类器。在第4章，我们曾经利用Logistic回归来预测患有疝病的马是否能够存活。而在本节，我们则想要知道如果利用多个单层决策树和AdaBoost能不能预测得更准。

> 示例：在一个难数据集上的AdaBoost应用
> (1) 收集数据：提供的文本文件。
> (2) 准备数据：确保类别标签是+1和□1而非1和0。
> (3) 分析数据：手工检查数据。
> (4) 训练算法：在数据上，利用adaBoostTrainDS()函数训练出一系列的分类器。
> (5) 测试算法：我们拥有两个数据集。在不采用随机抽样的方法下，我们就会对AdaBoost和Logistic回归的结果进行完全对等的比较。
> (6) 使用算法：观察该例子上的错误率。不过，也可以构建一个Web网站，让驯马师输入马的症状然后预测马是否会死去。

《机器学习实战》学习笔记（五）：Logistic 回归中，我们使用了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：

```
正确率:73.134328%
```

使用的是 `sklearn.linear_model.LogisticRegression` 函数进行训练的分类器，可以看到正确率如上图，如果使用我们的AdaBoost元算法呢?

```python
import numpy as np
import matplotlib.pyplot as plt

def loadDataSet(fileName):
    numFeat = len((open(fileName).readline().split('\t')))
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = []
        curLine = line.strip().split('\t')
        for i in range(numFeat - 1):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat, labelMat

"""
Parameters:
    dataMatrix - 数据矩阵
    dimen - 第dimen列，也就是第几个特征
    threshVal - 阈值
    threshIneq - 标志
Returns:
    retArray - 分类结果
"""
# 单层决策树分类函数
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = np.ones((np.shape(dataMatrix)[0],1))        #初始化retArray为1
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0 #如果小于阈值,则赋值为-1
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0  #如果大于阈值,则赋值为-1
    return retArray

"""
Parameters:
    dataArr - 数据矩阵
    classLabels - 数据标签
    D - 样本权重
Returns:
    bestStump - 最佳单层决策树信息
    minError - 最小误差
    bestClasEst - 最佳的分类结果
"""
# 找到数据集上最佳的单层决策树
```

```python
def buildStump(dataArr,classLabels,D):
    dataMatrix = np.mat(dataArr); labelMat = np.mat(classLabels).T
    m,n = np.shape(dataMatrix)
    numSteps = 10.0; bestStump = {}; bestClasEst = np.mat(np.zeros((m,1)))
    minError = float('inf')                                          #最小误差初始化为正无穷大
    for i in range(n):                                               #遍历所有特征
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max()      #找到特征中最小的值和最大值
        stepSize = (rangeMax - rangeMin) / numSteps                  #计算步长
        for j in range(-1, int(numSteps) + 1):
            for inequal in ['lt', 'gt']:                             #大于和小于的情况，均遍历。lt:less than, gt:
                threshVal = (rangeMin + float(j) * stepSize)         #计算阈值
                predictedVals = stumpClassify(dataMatrix, i, threshVal, inequal)#计算分类结果
                errArr = np.mat(np.ones((m,1)))                      #初始化误差矩阵
                errArr[predictedVals == labelMat] = 0                #分类正确的,赋值为0
                weightedError = D.T * errArr                         #计算误差
                # print("split: dim %d, thresh %.2f, thresh ineqal: %s, the weighted error is %.3f" % (i, threshVal, ineq
                if weightedError < minError:                         #找到误差最小的分类方式
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump, minError, bestClasEst

"""
Parameters:
    dataArr - 数据矩阵
    classLabels - 数据标签
    numIt - 最大迭代次数
Returns:
    weakClassArr - 训练好的分类器
    aggClassEst - 类别估计累计值
"""
# 使用AdaBoost算法提升弱分类器性能
def adaBoostTrainDS(dataArr, classLabels, numIt = 40):
    weakClassArr = []
    m = np.shape(dataArr)[0]
    D = np.mat(np.ones((m, 1)) / m)                                  #初始化权重
    aggClassEst = np.mat(np.zeros((m,1)))
    for i in range(numIt):
        bestStump, error, classEst = buildStump(dataArr, classLabels, D)  #构建单层决策树
        # print("D:",D.T)
        alpha = float(0.5 * np.log((1.0 - error) / max(error, 1e-16)))    #计算弱学习算法权重alpha,使error不等于0,因为分母不
        bestStump['alpha'] = alpha                                   #存储弱学习算法权重
        weakClassArr.append(bestStump)                               #存储单层决策树
        # print("classEst: ", classEst.T)
        expon = np.multiply(-1 * alpha * np.mat(classLabels).T, classEst) #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
        #计算AdaBoost误差，当误差为0的时候，退出循环
        aggClassEst += alpha * classEst                              #计算类别估计累计值
        # print("aggClassEst: ", aggClassEst.T)
        aggErrors = np.multiply(np.sign(aggClassEst) != np.mat(classLabels).T, np.ones((m,1)))#计算误差
        errorRate = aggErrors.sum() / m
        # print("total error: ", errorRate)
        if errorRate == 0.0: break                                   #误差为0，退出循环
    return weakClassArr, aggClassEst

"""
Parameters:
    datToClass - 待分类样例
    classifierArr - 训练好的分类器
Returns:
    分类结果
"""
# AdaBoost分类函数
def adaClassify(datToClass,classifierArr):
    dataMatrix = np.mat(datToClass)
    m = np.shape(dataMatrix)[0]
    aggClassEst = np.mat(np.zeros((m,1)))
    print(len(classifierArr))
```

```
    for i in range(len(classifierArr)):                                    #遍历所有分类器，进行分类
        classEst = stumpClassify(dataMatrix, classifierArr[i]['dim'], classifierArr[i]['thresh'], classifierArr[i]['ineq'
        aggClassEst += classifierArr[i]['alpha'] * classEst
        # print(aggClassEst)
    return np.sign(aggClassEst)


if __name__ == '__main__':
    dataArr, LabelArr = loadDataSet('horseColicTraining2.txt')
    weakClassArr, aggClassEst = adaBoostTrainDS(dataArr, LabelArr)
    testArr, testLabelArr = loadDataSet('horseColicTest2.txt')
    print(weakClassArr)
    predictions = adaClassify(dataArr, weakClassArr)
    errArr = np.mat(np.ones((len(dataArr), 1)))
    print('训练集的错误率:%.3f%%' % float(errArr[predictions != np.mat(LabelArr).T].sum() / len(dataArr) * 100))
    predictions = adaClassify(testArr, weakClassArr)
    errArr = np.mat(np.ones((len(testArr), 1)))
    print('测试集的错误率:%.3f%%' % float(errArr[predictions != np.mat(testLabelArr).T].sum() / len(testArr) * 100))
```

这里输出了AdaBoost 算法训练好的分类器组合

```
[{'dim': 9, 'thresh': 3.0,
40
训练集的错误率:19.732%
40
测试集的错误率:19.403%
```

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Logistic 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。

不同弱分类器数目情况下的AdaBoost测试和分类错误率。该数据集是个难数据集。通常情况下，AdaBoost会达到一个稳定的测试错误率，而并不会随分类器数目的增多而提高

分类器数目	训练错误率（%）	测试错误率（%）
1	0.28	0.27
10	0.23	0.24
50	0.19	0.21
100	0.19	0.22
500	0.16	0.25
1000	0.14	0.31
10000	0.11	0.33

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，如上表，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，而对测试集分类效果差，这是我们不希望看到的。

很多人都认为，AdaBoost 和SVM 是监督机器学习中最强大的两种方法。实际上，这两者之间拥有不少相似之处。我们可以把弱分类器想象成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。而它们的不同就在于其所定义的间隔计算方式有所不同，因此导致的结果也不同。特别是在高维空间下，这两者之间的差异就会更加明显。

6、Sklearn构建AdaBoost 回归分类器

```
import numpy as np
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

def loadDataSet(fileName):
    numFeat = len((open(fileName).readline().split('\t')))
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = []
        curLine = line.strip().split('\t')
```

```
        for i in range(numFeat - 1):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat, labelMat


if __name__ == '__main__':
    dataArr, classLabels = loadDataSet('horseColicTraining2.txt')
    testArr, testLabelArr = loadDataSet('horseColicTest2.txt')
    bdt = AdaBoostClassifier(DecisionTreeClassifier(max_depth = 2), algorithm = "SAMME", n_estimators = 10)
    bdt.fit(dataArr, classLabels)
    predictions = bdt.predict(dataArr)
    errArr = np.mat(np.ones((len(dataArr), 1)))
    print('训练集的错误率:%.3f%%' % float(errArr[predictions != classLabels].sum() / len(dataArr) * 100))
    predictions = bdt.predict(testArr)
    errArr = np.mat(np.ones((len(testArr), 1)))
    print('测试集的错误率:%.3f%%' % float(errArr[predictions != testLabelArr].sum() / len(testArr) * 100))
```

```
训练集的错误率:16.054%
测试集的错误率:17.910%
```

我们使用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，你会发现跟我们自己写的代码，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。

7、sklearn.ensemble.AdaBoostClassifier

sklearn.ensemble.AdaBoostClassifier是一个很好的模型，决策树算法就是通过它实现的，详细的看这个博客——sklearn.ensemble.AdaBoostClassifier()函数解析（最清晰的解释）

8、分类器性能评价

在结束分类这个主题之前，还必须讨论一个问题。在前面六章的所有分类介绍中（k-近邻算法、决策树、基于概率论的分类方法：朴素贝叶斯、Logistic 回归和支持向量机），我们都假设所有类别的分类代价是一样的。例如在第5章：《机器学习实战》学习笔记（五）：Logistic 回归，构建了一个用于检测患疝病的马匹是否存活的系统。在那里，我们构建了分类器，但是并没有对分类后的情形加以讨论。

- 假如某人给我们牵来一匹马，他希望我们能预测这匹马能否生存。我们说马会死，那么他们就可能会对马实施安乐死，而不是通过给马喂药来延缓其不可避免的死亡过程。我们的预测也许是错误的，马本来是可以继续活着的。毕竟，我们的分类器只有80%的精确率（accuracy）。如果我们预测错误，那么我们将会错杀了一个如此昂贵的动物，更不要说人对马还存在情感上的依恋。
- 如何过滤垃圾邮件呢？如果收件箱中会出现某些垃圾邮件，但合法邮件永远不会扔进垃圾邮件夹中，那么人们是否会满意呢？
- 癌症检测又如何呢？只要患病的人不会得不到治疗，那么再找一个医生来看看会不会更好呢（即情愿误判也不漏判）？

还可以举出很多很多这样的例子，坦白地说，在大多数情况下不同类别的分类代价并不相等，这就是非均衡分类问题。我们将会考察一种新的分类器性能度量方法，而不再是简单的通过错误率进行评价，并通过图像技术来对在上述非均衡问题下不同分类器的性能进行可视化处理。然后，考察这两种分类器的变换算法，它们能够将不同决策的代价考虑在内。

1）分类性能度量指标：正确率、召回率及ROC 曲线

我们在 《机器学习》周志华西瓜书学习笔记（二）：模型评估与选择 都有过详细的介绍，这里就简单说一下。

到现在为止，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例中错分的样例比例。**实际上，这样的度量错误掩盖了样例如何被分错的事实。在机器学习中，有一个普遍适用的称为 **混淆矩阵（confusion matrix）** 的工具，它可以帮助人们更好地了解分类中的错误。有这样一个关于在房子周围可能发现的动物类型的预测，这个预测的三类问题的混淆矩阵如表所示。

		预测结果		
		狗	猫	鼠
真实结果	狗	24	2	5
	猫	2	27	0
	鼠	4	2	30

利用混淆矩阵就可以更好地理解分类中的错误了。如果矩阵中的非对角元素均为0，就会得到一个完美的分类器。

接下来，我们考虑另外一个混淆矩阵，这次的矩阵只针对一个简单的二类问题。在下表中，给出了该混淆矩阵。在这个二类问题中，如果将一个正例判为正例，那么就可以认为产生了一个 **真正例（True Positive，TP，也称真阳）**；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（True Negative，TN，也称真阴）**。相应地，另外两种情况则分别称为 **伪反例（False Negative，FN，也称假阴）** 和 **伪正例（False Positive，FP，也称假阳）**。这4种情况如下表所示。

		预测结果	
		+1	−1
真实结果	+1	真正例（TP）	伪反例（FN）
	−1	伪正例（FP）	真反例（TN）

在分类中，当某个类别的重要性高于其他类别时，我们就可以利用上述定义来定义出多个比错误率更好的新指标。第一个指标是正确率（Precision），它等于TP/(TP+FP)，给出的是预测为正例的样本中的真正正例的比例。第二个指标是召回率（Recall），它等于TP/(TP+FN)，给出的是预测为正例的真实正例占所有真实正例的比例。在召回率很大的分类器中，真正判错的正例的数目并不多。

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。构建一个同时使正确率和召回率最大的分类器是具有挑战性的。

另一个用于度量分类中的非均衡性的工具是 **ROC曲线（ROC curve）**，ROC代表 **接收者操作特征（receiver operating characteristic）**，它最早在二战期间由电气工程师构建雷达系统时使用过，如下图。

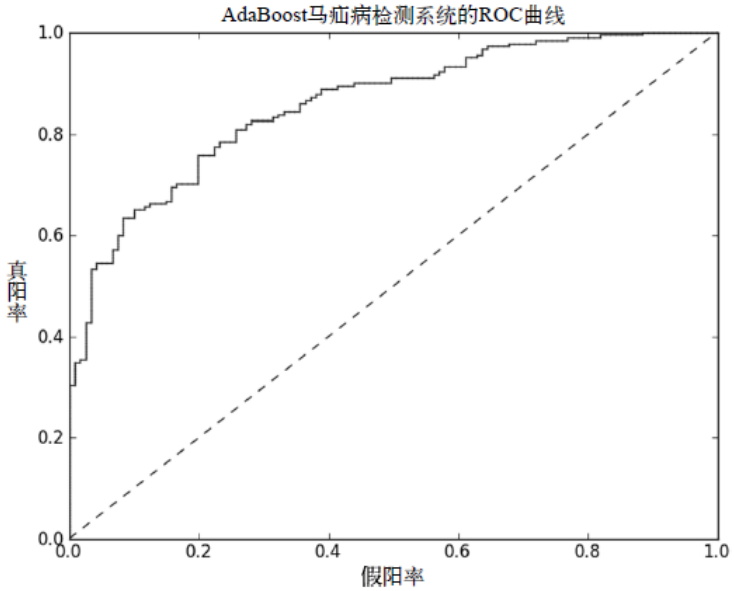


图7-3　利用10个单层决策树的AdaBoost马疝病检测系统的ROC曲线

在ROC曲线中，给出了两条线，一条虚线一条实线。图中的横轴是伪正例的比例（假阳率=FP/(FP+TN)），而纵轴是真正例的比例（真阳率=TP/(TP+FN)）。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。左下角的点所对应的是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。

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。由于在不同的阈值下，不同的分类器的表现情况可能各不相同，因此以某种方式将它们组合起来或许会更有意义。如果只是简单地观察分类器的错误率，那么我们就难以得到这种更深入的洞察效果了。

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。例如在垃圾邮件的过滤中，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为垃圾邮件而放入垃圾邮件的文件夹中。

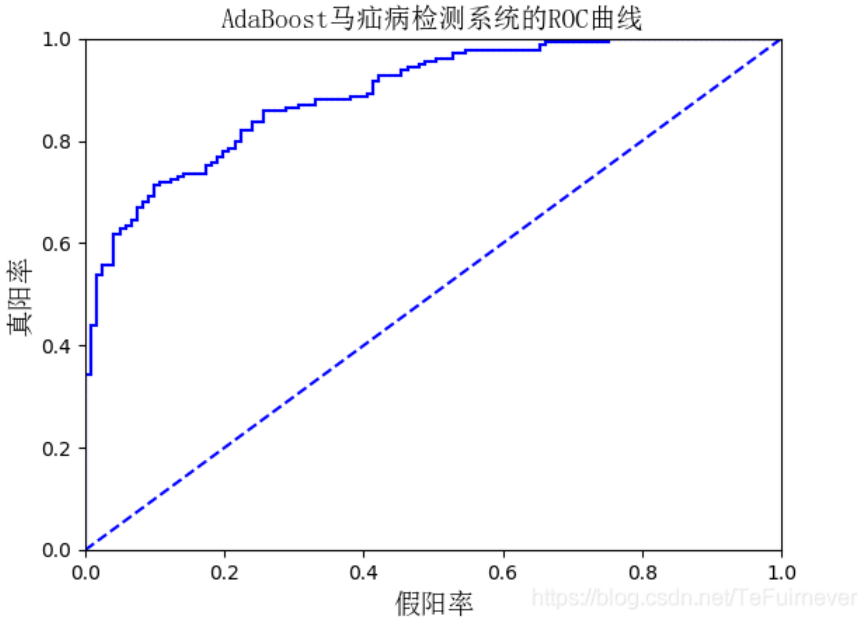
对不同的ROC曲线进行比较的一个指标是 **曲线下的面积（Area Unser the Curve，AUC）**。AUC给出的是分类器的平均性能值，当然它并不能完全代替对整条曲线的观察。一个完美分类器的AUC为1.0，而随机猜测的AUC则为0.5。

为了画出ROC曲线，分类器必须提供每个样例被判为阳性或者阴性的可信程度值。尽管大多数分类器都能做到这一点，但是通常情况下，这些值会在最后输出离散分类标签之前被清除。朴素贝叶斯能够提供一个可能性，而在Logistic回归中输入到Sigmoid函数中的是一个数值。在AdaBoost和SVM中，都会计算出一个数值然后输入到sign()函数中。所有的这些值都可以用于衡量给定分类器的预测强度。为了创建ROC曲线，首先要将分类样例按照其预测强度排序。先从排名最低的样例开始，所有排名更低的样例都被判为反例，而所有排名更高的样例都被判为正例。该情况的对应点为<1.0,1.0>。然后，将其移到排名次低的样例中去，如果该样例属于正例，那么对真阳率进行修改；如果该样例属于反例，那么对假阴率进行修改。

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

def loadDataSet(fileName):
    numFeat = len((open(fileName).readline().split('\t')))
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = []
        curLine = line.strip().split('\t')
        for i in range(numFeat - 1):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))

    return dataMat, labelMat

"""
Parameters:
    dataMatrix - 数据矩阵
    dimen - 第dimen列，也就是第几个特征
    threshVal - 阈值
    threshIneq - 标志
Returns:
    retArray - 分类结果
"""
# 单层决策树分类函数
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = np.ones((np.shape(dataMatrix)[0],1))          #初始化retArray为1
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0    #如果小于阈值,则赋值为-1
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0     #如果大于阈值,则赋值为-1
    return retArray

"""
Parameters:
    dataArr - 数据矩阵
    classLabels - 数据标签
    D - 样本权重
Returns:
    bestStump - 最佳单层决策树信息
    minError - 最小误差
    bestClasEst - 最佳的分类结果
"""
# 找到数据集上最佳的单层决策树
def buildStump(dataArr,classLabels,D):
    dataMatrix = np.mat(dataArr); labelMat = np.mat(classLabels).T
    m,n = np.shape(dataMatrix)
    numSteps = 10.0; bestStump = {}; bestClasEst = np.mat(np.zeros((m,1)))
    minError = float('inf')                                                    #最小误差初始化为正无穷大
    for i in range(n):                                                         #遍历所有特征
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max()     #找到特征中最小的值和最大值
        stepSize = (rangeMax - rangeMin) / numSteps                            #计算步长
        for j in range(-1, int(numSteps) + 1):
            for inequal in ['lt', 'gt']:                                       #大于和小于的情况，均遍历。lt:less than，gt:
                threshVal = (rangeMin + float(j) * stepSize)                   #计算阈值
                predictedVals = stumpClassify(dataMatrix, i, threshVal, inequal)#计算分类结果
                errArr = np.mat(np.ones((m,1)))                                #初始化误差矩阵
```

```
                errArr[predictedVals == labelMat] = 0                           #分类正确的,赋值为0
                weightedError = D.T * errArr                                    #计算误差
                # print("split: dim %d, thresh %.2f, thresh ineqal: %s, the weighted error is %.3f" % (i, threshVal, ineq
                if weightedError < minError:                                    #找到误差最小的分类方式
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump, minError, bestClasEst

"""
Parameters:
    dataArr - 数据矩阵
    classLabels - 数据标签
    numIt - 最大迭代次数
Returns:
    weakClassArr - 训练好的分类器
    aggClassEst - 类别估计累计值
"""
# 使用AdaBoost算法训练分类器
def adaBoostTrainDS(dataArr, classLabels, numIt = 40):
    weakClassArr = []
    m = np.shape(dataArr)[0]
    D = np.mat(np.ones((m, 1)) / m)                                             #初始化权重
    aggClassEst = np.mat(np.zeros((m,1)))
    for i in range(numIt):
        bestStump, error, classEst = buildStump(dataArr, classLabels, D) #构建单层决策树
        # print("D:",D.T)
        alpha = float(0.5 * np.log((1.0 - error) / max(error, 1e-16)))   #计算弱学习算法权重alpha,使error不等于0,因为分母不能
        bestStump['alpha'] = alpha                                              #存储弱学习算法权重
        weakClassArr.append(bestStump)                                          #存储单层决策树
        # print("classEst: ", classEst.T)
        expon = np.multiply(-1 * alpha * np.mat(classLabels).T, classEst)#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
        #计算AdaBoost误差，当误差为0的时候，退出循环
        aggClassEst += alpha * classEst                                         #计算类别估计累计值
        # print("aggClassEst: ", aggClassEst.T)
        aggErrors = np.multiply(np.sign(aggClassEst) != np.mat(classLabels).T, np.ones((m,1)))#计算误差
        errorRate = aggErrors.sum() / m
        # print("total error: ", errorRate)
        if errorRate == 0.0: break                                              #误差为0，退出循环
    return weakClassArr, aggClassEst

"""
Parameters:
    predStrengths - 分类器的预测强度
    classLabels - 类别
Returns:
    无
"""
# 绘制ROC
def plotROC(predStrengths, classLabels):
    font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)
    cur = (1.0, 1.0)                                    #绘制光标的位置
    ySum = 0.0                                          #用于计算AUC
    numPosClas = np.sum(np.array(classLabels) == 1.0)   #统计正类的数量
    yStep = 1 / float(numPosClas)                       #y轴步长
    xStep = 1 / float(len(classLabels) - numPosClas)    #x轴步长

    sortedIndicies = predStrengths.argsort()            #预测强度排序

    fig = plt.figure()
    fig.clf()
    ax = plt.subplot(111)
    for index in sortedIndicies.tolist()[0]:
        if classLabels[index] == 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], 'b--')
    plt.title('AdaBoost马疝病检测系统的ROC曲线', FontProperties = font)
    plt.xlabel('假阳率', FontProperties = font)
    plt.ylabel('真阳率', FontProperties = font)
    ax.axis([0, 1, 0, 1])
    print('AUC面积为:', ySum * xStep)                               #计算AUC
    plt.show()


if __name__ == '__main__':
    dataArr, LabelArr = loadDataSet('horseColicTraining2.txt')
    weakClassArr, aggClassEst = adaBoostTrainDS(dataArr, LabelArr)
    plotROC(aggClassEst.T, LabelArr)
```




```
AUC面积为： 0.8918191104095092
```

可以看到有两个输出结果，一个是AUC面积，另一个ROC曲线图，具体的我们在上面已经解释过了。

2）基于代价函数的分类器决策控制

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代价的方法，其中的一种称为 **代价敏感的学习（cost-sensitive learning）**。考虑表7-4中的代价矩阵，第一张表给出的是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：$TP*0+FN*1+FP*1+TN*0$。

接下来我们考虑下面的第二张表，基于该代价矩阵的分类代价的计算公式为：$TP*(-5)+FN*1+FP*50+TN*0$。采用第二张表作为代价矩阵时，两种分类错误的代价是不一样的。类似地，这两种正确分类所得到的收益也不一样。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。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。

表7-4　一个二类问题的代价矩阵

		预测结果	
		+1	−1
真实结果	+1	0	1
	−1	1	0

		预测结果	
		+1	−1
真实结果	+1	−5	1
	−1	50	0

3）处理非均衡问题的数据抽样方法

另外一种针对非均衡问题调节分类器的方法，就是对分类器的训练数据进行改造。这可以通过欠抽样（undersampling）或者过抽样（oversampling）来实现。过抽样意味着复制样例，而欠抽样意味着删除样例。不管采用哪种方式，数据都会从原始形式改造为新形式。抽样过程则可以通过随机方式或者某个预定方式来实现。

通常也会存在某个罕见的类别需要我们来识别，比如在信用卡欺诈当中。如前所述，正例类别属于罕见类别。我们希望对于这种罕见类别能尽可能保留更多的信息，因此，我们应该保留正例类别中的所有样例，而对反例类别进行欠抽样或者样例删除处理。这种方法的一个缺点就在于要确定哪些样例需要进行剔除。但是，在选择剔除的样例中可能携带了剩余样例中并不包含的有价值信息。

上述问题的一种解决办法，就是选择那些离决策边界较远的样例进行删除。假定我们有一个数据集，其中有50例信用卡欺诈交易和5000例合法交易。如果我们想要对合法交易样例进行欠抽样处理，使得这两类数据比较均衡的话，那么我们就需要去掉4950个样例，而这些样例中可能包含很多有价值的信息。这看上去有些极端，因此有一种替代的策略就是使用反例类别的欠抽样和正例类别的过抽样相混合的方法。

要对正例类别进行过抽样，我们可以复制已有样例或者加入与已有样例相似的点。一种方法是加入已有数据点的插值点，但是这种做法可能会导致过拟合的问题。

9、总结

集成方法通过组合多个分类器的分类结果，获得了比简单的单分类器更好的分类结果。有一些利用不同分类器的集成方法，但是本章只介绍了那些利用同一类分类器的集成方法。多个分类器组合可能会进一步凸显出单分类器的不足，比如过拟合问题。如果分类器之间差别显著，那么多个分类器组合就可能会缓解这一问题。分类器之间的差别可以是算法本身或者是应用于算法上的数据的不同。

本章介绍的两种集成方法是bagging和boosting。在bagging中，是通过随机抽样的替换方式，得到了与原始数据集规模一样的数据集。而boosting在bagging的思路上更进了一步，它在数据集上顺序应用了多个不同的分类器。另一个成功的集成方法就是随机森林，但是由于随机森林不如AdaBoost流行，所以我们并没有对它进行介绍。

本章介绍了boosting方法中最流行的一个称为AdaBoost的算法。AdaBoost以弱学习器作为基分类器，并且输入数据，使其通过权重向量进行加权。在第一次迭代当中，所有数据都等权重。但是在后续的迭代当中，前次迭代中分错的数据的权重会增大。这种针对错误的调节能力正是AdaBoost的长处。本章以单层决策树作为弱学习器构建了AdaBoost分类器。实际上，AdaBoost函数可以应用于任意分类器，只要该分类器能够处理加权数据即可。AdaBoost算法十分强大，它能够快速处理其他分类器很难处理的数据集。

非均衡分类问题是指在分类器训练时正例数目和反例数目不相等（相差很大）。该问题在错分正例和反例的代价不同时也存在。本章不仅考察了一种不同分类器的评价方法——ROC曲线，还介绍了正确率和召回率这两种在类别重要性不同时，度量分类器性能的指标。本章介绍了通过过抽样和欠抽样方法来调节数据集中的正例和反例数目。另外一种可能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。

到目前为止，我们介绍了一系列强大的分类技术。本章是分类部分的最后一章，接下来我们将进入另一类监督学习算法——回归方法，这也将完善我们对监督方法的学习。回归很像分类，但是和分类输出标称型类别值不同的是，回归方法会预测出一个连续值。