### 1案例基本信息

1. 案例名称：
2. 案例使用到的基本理论知识点
3. 案例使用的平台、语言及库函数

平台：

语言：

库函数：

### 2案例设计方案

1. 案例描述
2. 案例创新点

### 3案例代码

1. 案例数据样例或数据集
2. 案例代码
3. 案例结果

### 样例

### 6.1.1案例基本信息

这里我们通过对Adaboost集成学习算法所用到的基本理论知识进行简单介绍，以便读者能够更深入的了解Adaboost集成学习。

1. 案例名称：Adaboost集成学习
2. 案例使用到的基本理论知识点

Adaboost是英文"Adaptive Boosting"（自适应增强）的缩写，它的自适应在于：前一个基本分类器被错误分类的样本的权值会增大，而正确分类的样本的权值会减小，并再次用来训练下一个基本分类器。同时，在每一轮迭代中，加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数才确定最终的强分类器。

注：弱分类器可以是不同种的分类器组合也可以是同种分类器组合

boosting，也被成为增强学习或者提升法，是一种重要的集成学习技术，能够将预测精度仅比随即猜测略高的弱学习器增强为预测精度高的强学习器，这在直接构造强学习器非常困难的情况下，为学习算法的设计提供了一种有效的新思路和新方法。

1. 案例使用的平台、语言及库函数

平台：vscode

语言：Python

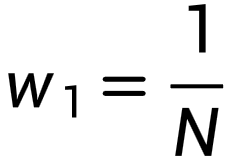
库函数：matplotlib、numpy

### 6.1.2案例设计方案

本小节主要对Adaboost集成学习算法的步骤以及其创新点进行介绍。

1. 案例描述

首先，是初始化训练数据的权值分布D1。假设有N个训练样本数据，则每一个训练样本最开始时，都被赋予相同的权值：

 （6.1）

然后，训练弱分类器hi。具体训练过程中是：如果某个训练样本点，被弱分类器hi准确地分类，那么在构造下一个训练集中，它对应的权值要减小（分类错的后面迭代的时候重点训练）；相反，如果某个训练样本点被错误分类，那么它的权值就应该增大。权值更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。

最后，将各个训练得到的弱分类器组合成一个强分类器。各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用，而降低分类误差率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。

换而言之，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

AdaBoost算法步骤：

首先，初始化训练数据的权值分布。

然后，训练弱分类器。具体训练过程中，如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它的权重就被降低；相反，如果某个样本点没有被准确地分类，那么它的权重就得到提高。然后，权重更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。

最后，将各个训练得到的弱分类器组合成强分类器。迭代完成后，最后的分类器是由迭代过程中选择的弱分类器线性加权得到的，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。案例技术路线图如图6-1所示。

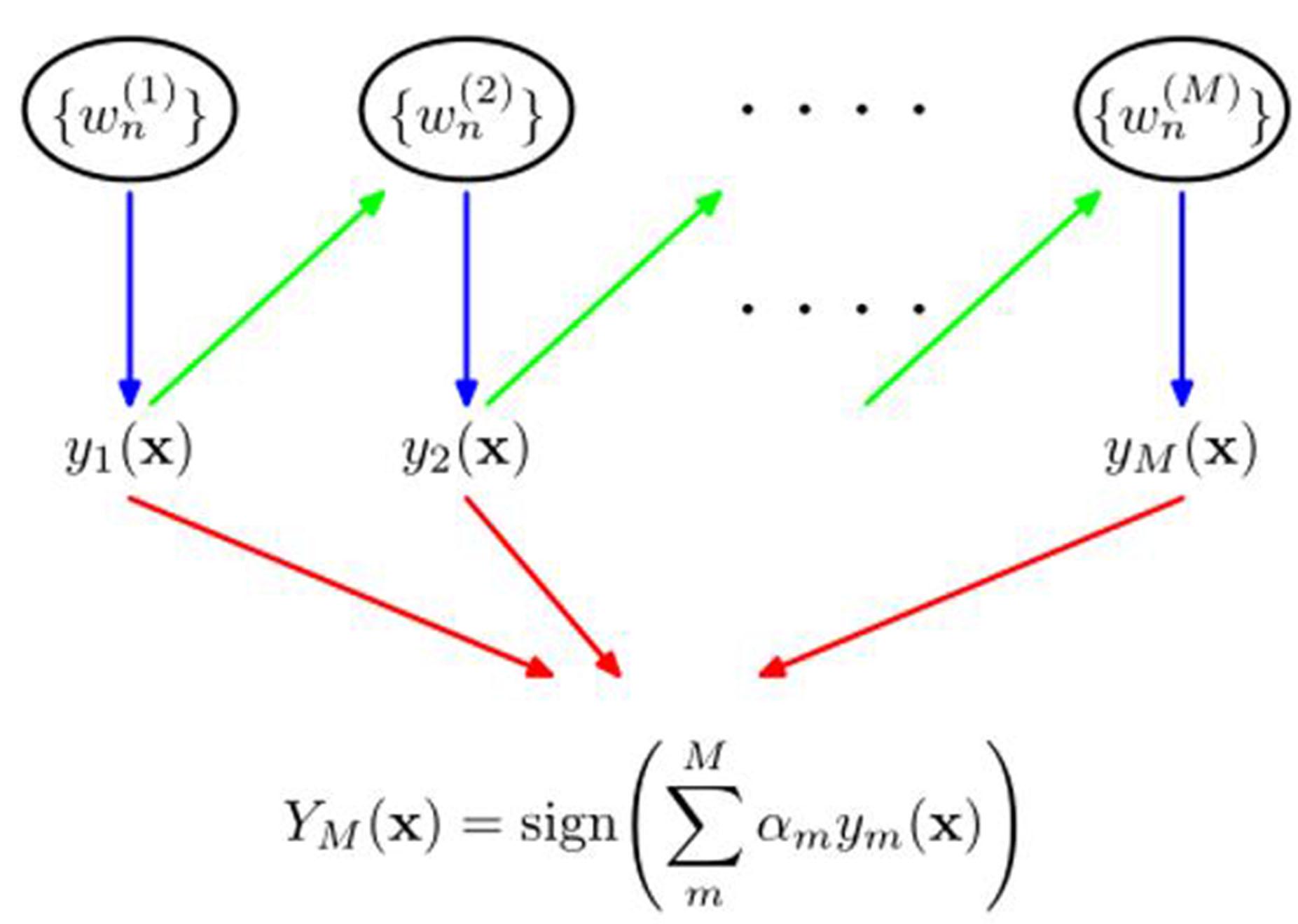


图6-1Adaboost案例技术路线图

1. 案例创新点

本案例中，主要通过随机设置5个数据集datMat，并分别带有2个特征，用Adaboost集成学习算法对这5个数据进行分类，并将最后结果进行可视化。首先Adaboost不容易发生过拟合，由于其并没有限制弱学习器的种类，所以可以使用不同的学习算法来构建弱分类器，其次Adaboost具有很高的精度，参数少，实际应用中不需要调节太多的参数，相对于Bagging算法和Random Forest算法，Adaboost充分考虑每个分类器的权重。

### 6.1.3案例代码

这一小节我们随机设置了5个带有标签的数据集，并用Adaboost集成学习算法对这5个数据集进行分类处理。

1. 案例数据样例或数据集

随机设置5个数据集datMat，分别都有2个特征，classLabels是数据集的所属类别标签。以下即为训练所用数据。

特征矩阵

datMat = matrix([[1.0, 2.1],

[2.0, 1.1],

[1.3, 1.0],

[1.0, 1.0],

[2.0, 1.0]])

类别标签矩阵

classLabels = [1,0, 1.0, -1.0, -1.0, 1.0]

1. 案例代码

Adaboost代码中数据集通过随机设置5个带有标签的数据，用Adaboost模型进行分类，其中找到数据集上最佳的单层决策树——单层决策树是指只考虑其中一个特征，用该特征进行分类，x=阈值(向量只有一个x轴没有y轴，那么用一条垂直于x轴的线进行分类即可)

    例如本文例子，如果以第一列特征为基础，阈值选择X=1.3这条竖线，并设置>1.3的为反例，<1.3的为正例，这样就构造了一个二分类器。

其中每迭代一次生成一个弱分类器，最大迭代次数虽然设置的是40个，但若迭代过程中不满40次误差就为0时就可以停止迭代了，说明用不了40个弱分类器就可以完全正确分类。接下来通过案例来演示。

Adaboost代码如例6-1所示：

【例6-1】

1. import numpy as np
2. import matplotlib.pyplot as plt
3. def loadSimpData():
4. dataMat = np.matrix([[1., 2.1],
5. [1.5, 1.6],
6. [1.3, 1.],
7. [1., 1.],
8. [2., 1.]])
9. classLabels = [1.0, 1.0, -1.0, -1.0, 1.0] # 正例1，反例-1
10. return dataMat, classLabels
11. #stumpClassify()函数是通过阈值比较对数据进行分类的。 所有在阈值一边的数据会分到类别-1, 而在另外一边的数据分到类别+l。（其中利用到了数组的过滤功能）
12. def stumpClassify(dataMatrix, dimen, threshVal, inequal):
13. retArray = np.ones((np.shape(dataMatrix)[0], 1)) # 初始化retArry为1(假设全为正例)
14. if inequal == 'lt':
15. retArray[dataMatrix[:, dimen] <= threshVal] = -1.0 # 如果小于阈值，则赋值为-1(反例) 注：此处的等号表示样本点恰在阈值线上，此处假设在此弱分类器中阈值线上的样本是反例(当然也可以设为正例，但是要注意训练样本和测试样本的分类函数的这个等号要一致)
16. else:
17. retArray[dataMatrix[:, dimen] > threshVal] = -1.0 # 如果大于阈值，则赋值为-1
18. return retArray
19. def buildStump(dataArr, classLabels, D):
20. dataMatrix = np.mat(dataArr)
21. labelMat = np.mat(classLabels).T # 将列表转换为向量
22. m, n = np.shape(dataMatrix)
23. numSteps = 10.0 # 总步数，计算步长用的
24. bestStump = {} # 用来保存单个最优弱分类器的信息的(第几个特征，分类的阈值，lt还是gt，此弱分类器的权重alpha)
25. bestClasEst = np.mat(np.zeros((m, 1))) # 保存最佳的分类结果
26. minOverallError = float('inf') # 最小总误差初始化为正无穷大
27. for i in range(n): # 分别对每个特征计算最优的划分阈值(分别对每个特征求其最小的总误差，得到最小总误差最小的那个特征，此特征被选为分类特征)
28. rangeMin = dataMatrix[:, i].min() # 每一行的第i个元素中最小的元素
29. rangeMax = dataMatrix[:, i].max() # 找到特征中最小和最大的值
30. stepSize = (rangeMax - rangeMin) / numSteps # 计算步长---阈值递增的步长
31. for j in range(-1, int(numSteps) + 1): # 计算阈值取各个值时的误差，找误差最小的那个阈值 j取(-1, int(numSteps) + 1)可以看作第几步(总共10步)，且方便下面的阈值计算
32. # lt是指在该阈值下，如果<阈值，则分类为-1
33. # gt是指在该阈值下,如果>阈值，则分类为-1，则是单层决策树分类算法，其中就这个题目来说，两者加起来误差肯定为1
34. # 再说通俗一点，画出阈值那条线后，还有两种情况，一种是阈值线的右边是正例，左边是反例(lt)；另一种是阈值线的左边是正例，右边是反例(gt)，所以每个阈值要计算两种情况的误差
35. for inequal in ['lt', 'gt']: # 大于和小于的情况，均遍历。lt:less than，gt:greater than
36. threshVal = (rangeMin + float(j) \* stepSize) # 计算阈值（从0.9到2.0，步长为0.1，挨个计算误差）
37. predictedVals = stumpClassify(dataMatrix, i, threshVal, inequal) # 计算分类结果，即若以当前threshVal为阈值分类，那么此时的训练样本分类结果如何（1表示正例，-1表示反例）
38. errArr = np.mat(np.ones((m, 1))) # 初始化误差矩阵(不是保存误差的，而是用来保存哪些样本分类错误，哪些样本分类正确)
39. errArr[predictedVals == labelMat] = 0 # 若分类正确的,则记为0，否则记为1，下面乘个该样本的权重当作误差（列表之间可以直接判断相等，predictedVals == labelMat返回[[ True], [ True], [False], [False], [ True]]）
40. # 基于权重向量D而不是其他错误计算指标来评价分类器的，不同的分类器计算方法不同
41. overallError = D.T \* errArr # 计算所有样本的总误差--这里没有采用常规方法来评价这个分类器的分类准确率，而是乘上的权重（此处因为是mat矩阵，所以\*是向量的乘法）
42. print("第%d个特征, 阈值为%.2f, ineqal: %s, 该阈值的决策树对所有样本的总误差为%.3f" % (i, threshVal, inequal, overallError))
43. if overallError < minOverallError: # 找到总误差最小的分类方式--找到当前最好的弱分类器
44. minOverallError = overallError
45. bestClasEst = predictedVals.copy() # 保存该阈值的分类结果
46. bestStump['dim'] = i # 保存特征
47. bestStump['thresh'] = threshVal # 保存最优阈值
48. bestStump['ineq'] = inequal # 保存是lt还是gt
49. return bestStump, minOverallError, bestClasEst
50. def adaBoostTrainDS(dataArr, classLabels, numIt = 40):
51. weakClassifiterArr = [] # 保存多个训练好的弱学习器
52. m = np.shape(dataArr)[0] # 行数(样本个数)，此句可理解为np.shape(dataArr)返回的元组(m, n)中的第0个数m
53. D = np.mat(np.ones((m, 1)) / m) # 初始化每个样本的权重(均是1/m)
54. aggClassEst = np.mat(np.zeros((m, 1))) # 保存每一轮累加的投票值(初始化为0)，后面最终判断某一区域是正例还是反例要用(对加权投票套sign()函数)
55. for i in range(numIt):
56. bestStump, error, bestClasEst = buildStump(dataArr, classLabels, D) # 构建单个单层决策树
57. #print("D:",D.T)
58. 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，因为分母不能为0(注意此权重是弱学习器的权重而非是单个样本的权重)
59. bestStump['alpha'] = alpha # 存储弱学习算法权重
60. print("第%d次迭代中得到的最优单层决策树：第%d个特征, 阈值为%.2f, ineqal: %s, 该阈值的决策树对所有样本的总误差为%.3f, 此弱分类器的权重为：%.3f" % (i, bestStump['dim'], bestStump['thresh'], bestStump['ineq'], error, bestStump['alpha']))
61. weakClassifiterArr.append(bestStump) #存储单层决策树
62. #print("bestClasEst:"bestClasEst.T)
63. expon = np.multiply(-1 \* alpha \* np.mat(classLabels).T, bestClasEst) # 计算e的指数项
64. D = np.multiply(D, np.exp(expon))
65. D = D / D.sum() # 根据样本权重公式，更新样本权重
66. # 计算AdaBoost误差，当误差为0的时候，退出循环
67. aggClassEst += alpha \* bestClasEst # 加权投票（对加权投票值取sign函数就可以得到预测值），注意这里包括了目前已经训练好的每一个弱分类器
68. print("前{}个弱分类器得到的aggClassEst:{} ".format(i, aggClassEst.T))
69. aggErrors = np.multiply(np.sign(aggClassEst) != np.mat(classLabels).T, np.ones((m, 1))) # 计算误差 aggErrors向量中元素为1的表示分错的样本，为0的表示分对的样本
70. # np.sign(aggClassEst) != np.mat(classLabels).T也可以写ClassEst != np.mat(classLabels).T，表示分类错了则为true，分类对了则为false,自动转换成0和1，
71. errorRate = aggErrors.sum() / m # aggErrors.sum()就表示为总共有多少个样本分类错误
72. print("分错样本个数/样本总个数: ", errorRate)
73. if errorRate == 0.0: break # 误差为0，说明样本被完全正确的分类了，不再需要更多的弱学习器了，退出循环
74. return weakClassifiterArr
75. def showDataSet(dataMat, labelMat, weakClassifiterArr):
76. data\_plus = [] # 正样本
77. data\_minus = [] # 负样本
78. for i in range(len(dataMat)): # 正负样本分类
79. if labelMat[i] > 0:
80. data\_plus.append(dataMat[i])
81. else:
82. data\_minus.append(dataMat[i])
83. data\_plus\_np = np.array(data\_plus) # 转换为numpy矩阵
84. data\_minus\_np = np.array(data\_minus)
85. # 绘制样本
86. plt.scatter(np.transpose(data\_plus\_np)[0], np.transpose(data\_plus\_np)[1],c='red') # 正样本 np.transpose(data\_plus\_np)[0]表示data\_plus\_np转置后的第0行
87. plt.scatter(np.transpose(data\_minus\_np)[0], np.transpose(data\_minus\_np)[1], c='green') # 负样本
88. # 绘制训练函数图像
89. for i in range(len(weakClassifiterArr)): # 每个弱分类器一条线(一个阈值)
90. if weakClassifiterArr[i]['dim'] == 0: # 如果分类特征是第0个特征(x1)
91. x2 = np.arange(1.0, 3.0, 1) # x2是一个2维列表[1,2]
92. plt.plot([weakClassifiterArr[i]['thresh'], weakClassifiterArr[i]['thresh']], x2) # 因为确定一条线至少要两个点，所以至少都是二维列表
93. else: # 如果分类特征是第1个特征(x2)
94. x1 = np.arange(1.0, 3.0, 1)
95. plt.plot(x1, [weakClassifiterArr[i]['thresh'], weakClassifiterArr[i]['thresh']])
96. plt.title('Training sample data') # 绘制title
97. # 绘制坐标轴
98. plt.xlabel('x1'); # 第0个特征
99. plt.ylabel('x2') # 第1个特征
100. plt.show()
101. def adaClassify(testSample,weakClassifiterArr):
102. dataMatrix = np.mat(testSample)
103. m = np.shape(dataMatrix)[0]
104. aggClassEst = np.mat(np.zeros((m,1)))
105. for i in range(len(weakClassifiterArr)): #遍历所有分类器，进行分类
106. bestClasEst = stumpClassify(dataMatrix, weakClassifiterArr[i]['dim'], weakClassifiterArr[i]['thresh'], weakClassifiterArr[i]['ineq'])
107. aggClassEst += weakClassifiterArr[i]['alpha'] \* bestClasEst #加权投票
108. print("测试样本的加权投票为：", aggClassEst)
109. return np.sign(aggClassEst)
110. if \_\_name\_\_ =='\_\_main\_\_':
111. dataArr, classLabels = loadSimpData() #返回训练样本
112. weakClassifiterArr = adaBoostTrainDS(dataArr, classLabels) #通过adaboost得到多个弱分类器，保存在weakClassifiterArr列表中
113. showDataSet(dataArr, classLabels,weakClassifiterArr) #画图
114. print(adaClassify([[1.3, 1.], [0, 0],[5, 5]], weakClassifiterArr)) #测试样进行测试
115. 案例结果

如图6-2中结果所示，每个弱分类器是一条线，且横轴x1是第0个特征，纵轴x2是第1个特征，遍历所有分类器，并用加权投票法来进行分类，最终将结果画图进行可视化。

运行结果如图6-2所示。

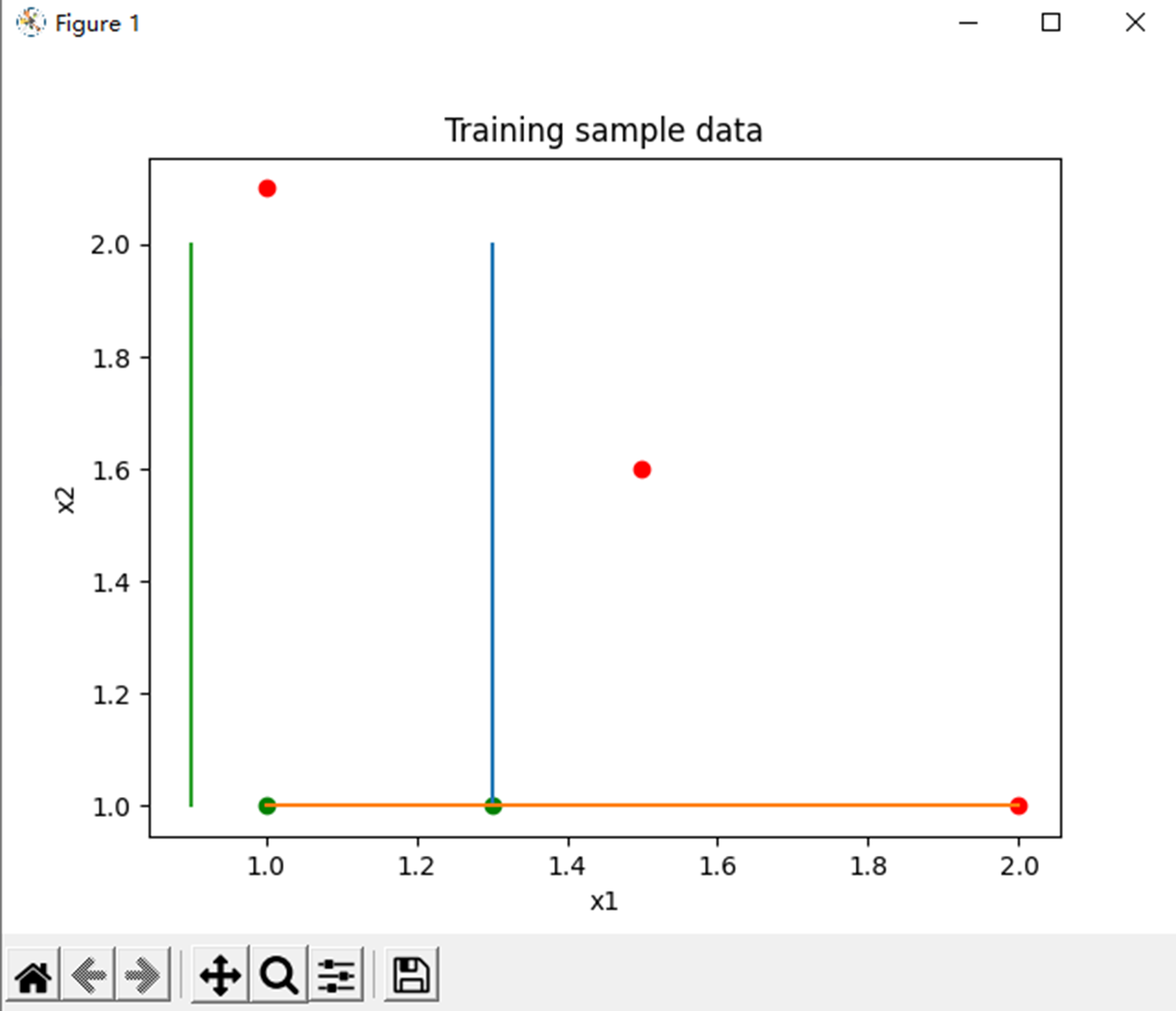


图6-2 Adaboost案例运行结果

如图6-3中结果显示，lt是指在该阈值下，如果<阈值，则分类为-1，gt是指在该阈值下,如果>阈值，则分类为-1，则是单层决策树分类算法，且两者加起来误差肯定为1。再通俗一点，画出阈值那条线后，还有两种情况，一种是阈值线的右边是正例，左边是反例(lt)；另一种是阈值线的左边是正例，右边是反例(gt)，所以每个阈值要计算两种情况的误差。

部分运行结果如图6-3所示。

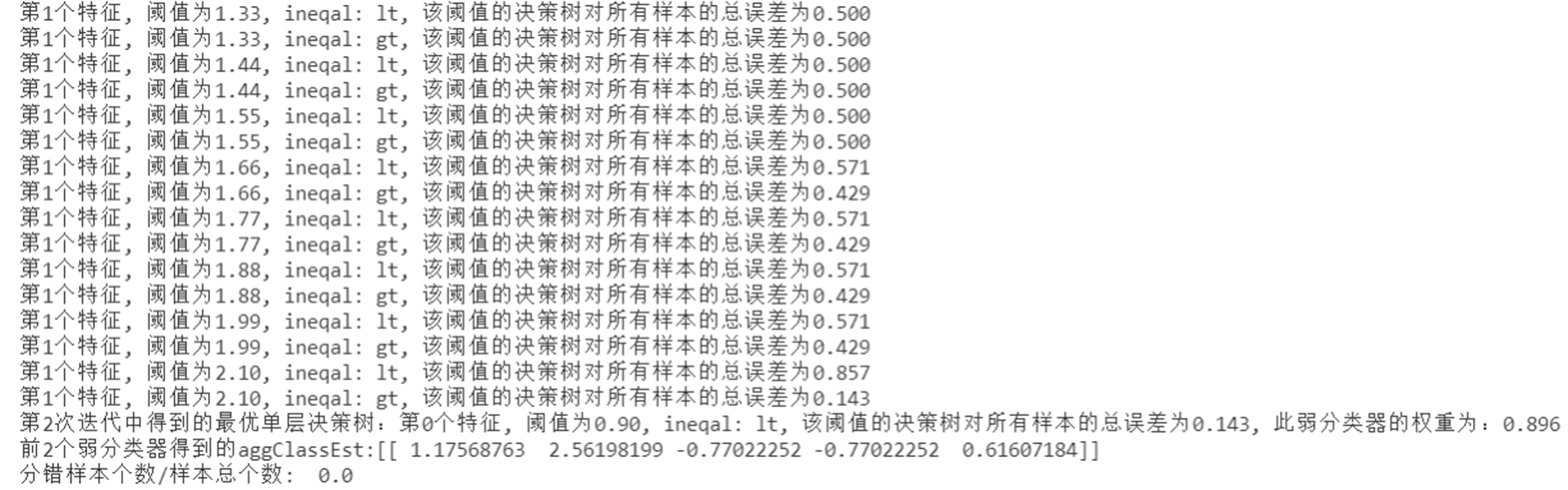


图6-3 Adaboost案例部分运行结果