机器学习第十二次作业

1. 算法实现

```
def loadSimpData():
    '''
    :return datMat: 密度、含糖率矩阵
    :return classLabels:  分类标签
    '''
    melon_data = pd.read_excel('C:/Users/mi/Desktop/melon.xlsx')
    datMat = np.matrix(melon_data[['密度','含糖率']])
    classLabels = melon_data['好瓜'].copy()
    classLabels[classLabels == '是'] = 1.0
    classLabels[classLabels == '否'] = -1.0
    return datMat,classLabels


def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    '''
    通过阈值比较对数据进行分类
    阈值两侧被分为-1和1类
    满足不等式条件，设为-1
    :param dataMatrix: 特征矩阵
    :param dimen: 特征的整数索引
    :param threshVal: 阈值
    :param threshIneq: 不等式条件 'lt' or 'gt'
    :return retArray: 返回数组
    '''
    retArray = np.ones((np.shape(dataMatrix)[0],1))
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0
    return retArray

def buildStump(dataArr,classLabels,D):
    '''
    遍历stumpClassify()函数所有的可能输入值，并基于数据的权重向量D找到数
    据集上最佳的单层决策树
    :param dataArr: 数据集
    :param classLabels: 类别标签
    :param D:样本权重向量
    :return bestStump: 分类错误率最低的单层决策树
    :return minError: 最小错误率
    :return bestClasEst:
    '''
    dataMatrix = np.matrix(dataArr);
    labelMat = np.matrix(classLabels).T
    m,n = np.shape(dataMatrix)
    # 在特征的所有可能值上进行遍历
    numSteps = m*n
    # 用字典存储给定权重向量D时所得到的最佳单层决策树
    bestStump = {}
```

```
    bestClasEst = np.matrix(np.zeros((m,1)))
    # 初始化为正无穷
    minError = np.inf
    # 在数据集的所有特征上遍历
    for i in range(n):
        # 计算特征的最大值与最小值来了解阈值的遍历步长
        rangeMin = min(dataMatrix[:,i])
        rangeMax = max(dataMatrix[:,i])
        stepSize = (rangeMax-rangeMin)/numSteps
        # 遍历所有阈值
        for j in range(-1,int(numSteps)+1):
            # 在大于和小于之间切换不等式
            for inequal in ['lt','gt']:
                # 得到遍历的阈值
                threshVal = (rangeMin + float(j) * stepSize)
                # 获得预测结果
                predictedVals = stumpClassify(dataMatrix,i,threshVal,inequal)
                # 如果预测结果与真实结果不同，errArr对应位置记为1
                errArr = np.matrix(np.ones((m,1)))
                errArr[predictedVals == labelMat] = 0
                # 结合权重向量D计算集成错误率
                weightedError = D.T * errArr
                print("split: dim %d, thresh %.2f, thresh ineqal: %s, the
weighted error is %.3f" %(i, threshVal, inequal, weightedError))
                # 比较当前错误率和最小错误率，若小则更新错误率并保留单层决策树
                if weightedError < minError:
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump,minError,bestClasEst

def adaBoostTrainDS(dataArr,classLabels,numIt=40):
    '''
    :param dataArr: 数据集
    :param classLabels: 类别标签
    :param numIt=40: 最大迭代次数(分类器个数)
    :return weakClassArr: 字典列表
    '''
    weakClassArr = []
    m = np.shape(dataArr)[0]
    D = np.matrix(np.ones((m,1))/m)
    # 记录类别估计值
    aggClassEst = np.matrix(np.zeros((m,1)))
    for i in range(numIt):
        bestStump,error,classEst = buildStump(dataArr,classLabels,D)
        print("D:",D.T)
        # 计算alpha值
        alpha = float(0.5*np.log((1.0-error)/max(error,1e-16)))
        # 记录alpha值
        bestStump['alpha'] = alpha
        weakClassArr.append(bestStump)
        print("classEst:",classEst.T)
        expon =
np.multiply(-1*alpha*np.matrix(classLabels,dtype=float).T,classEst)
        D = np.multiply(D,np.exp(expon))
        D = D/D.sum()
```

```
        # 调整样本分布
        aggClassEst += alpha*classEst
        print("aggClassEst:",aggClassEst.T)
        aggErrors =
np.multiply(np.sign(aggClassEst.T)!=np.matrix(classLabels).T,np.ones((m,1)))
        errorRate = sum(aggErrors)/m
        print("total error:",errorRate,"\n")
        # 训练错误为0则提前结束循环
        if errorRate.any() == 0.0: break
    return weakClassArr
```

2. 模块调用与字体设置

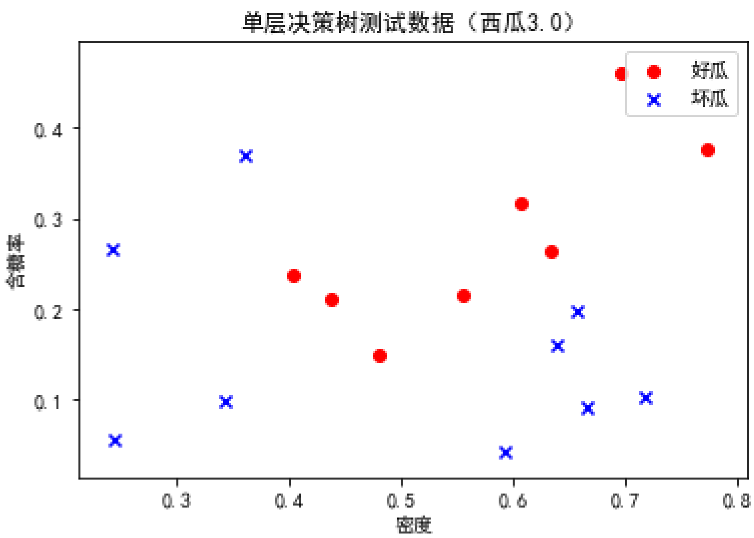
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

3. 主函数部分

```
# 读取数据
datMat,classLabels = loadSimpData()

# 绘制训练数据散点图
plot1 = plt.figure(1)
plt.title('单层决策树测试数据（西瓜3.0）')
plt.scatter(datMat[classLabels == 1,0].tolist(),datMat[classLabels ==
1,1].tolist(),marker='o',color ='r',label='好瓜')
plt.scatter(datMat[classLabels == -1,0].tolist(),datMat[classLabels ==
-1,1].tolist(),marker='x',color ='b',label='坏瓜')
plt.xlabel('密度')
plt.ylabel('含糖率')
plt.legend(loc='upper right')
```



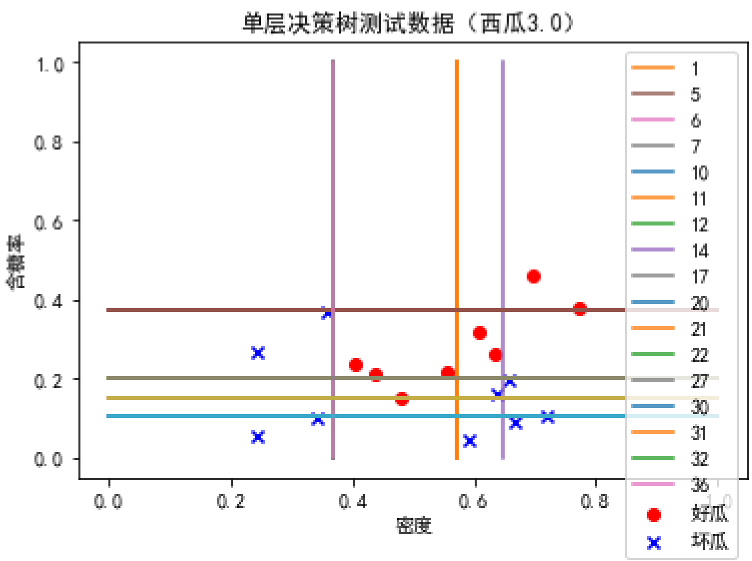
```python
# 测试函数
D = np.matrix(np.ones((17,1))/17)
bestStump,minError,bestClasEst = buildStump(datMat,classLabels,D)

# 设置最大训练轮数（基学习器个数）
iter_max = 40
# 集成学习
classifierArray = adaBoostTrainDS(datMat,classLabels,iter_max)
```

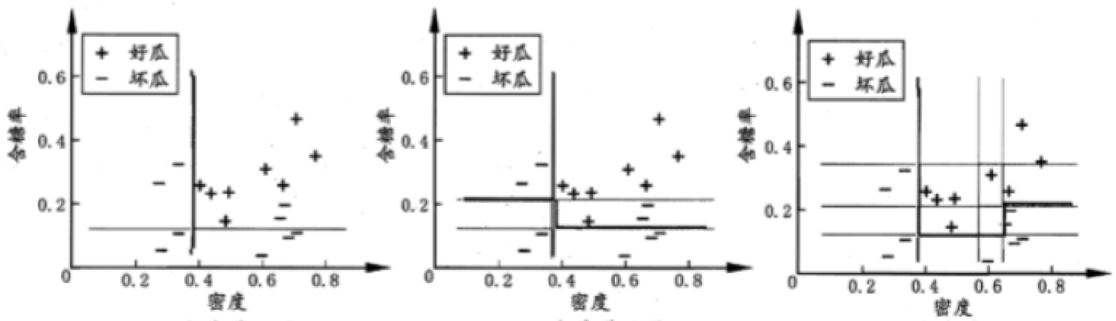
```python
# 绘制不同数量弱训练器生成的决策边界
plot2 = plt.figure(2)
plt.title('单层决策树测试数据（西瓜3.0）')
plt.scatter(datMat[classLabels == 1,0].tolist(),datMat[classLabels ==
1,1].tolist(),marker='o',color ='r',label='好瓜')
plt.scatter(datMat[classLabels == -1,0].tolist(),datMat[classLabels ==
-1,1].tolist(),marker='x',color ='b',label='坏瓜')
plt.xlabel('密度')
plt.ylabel('含糖率')

for i in range(iter_max):
    thresh = float(classifierArray[i]['thresh'])
    if classifierArray[i]['dim'] == 0:
        plt.plot([thresh,thresh],[0,1],label= i)
    else:
        plt.plot([0,1],[thresh,thresh])

plt.legend(loc='upper right')
```



对比书上的结论



(a) 3个基学习器　(b) 5个基学习器　(c) 11个基学习器

图 8.4　西瓜数据集 3.0α 上 AdaBoost 集成规模为 3、5、11 时, 集成(红色)与基学习器(黑色)的分类边界.