机器学习——提升分类器性能

在一般的分类器模型中，它们的分类正确率是有限的，如果我们能够通过一定的集成方法实现多种学习器的集合，就可以实现强可学习，提高学习能力，提升正确率。实现集成的方法有Bagging和Boosting。

Bagging

Bagging叫做自举汇聚法，通过对训练数据采用自举采样，放回性地采集数据，主要思想如下：

首先原始样本集中抽取训练集，然后每一轮从原始样本集中使用Bagging的方法抽取n个训练样本，也就是说如果我们抽了k次，就得到k个训练集，这些训练集是互相独立的。然后我们对于每一个训练集训练得到k个模型，基于这k个模型采用投票的方式得到分类结果。如果是回归问题，我们就需要根据得到的参数取平均值来得到最后的结果。

Boosting

Boosting的方法是使用重赋权迭代训练基分类器，主要思想如下：

每一次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于上次的分类结果，而基分类器采用序列性的线性加权进行组合。

总的来说，Boosting的训练集在每一轮是不变的，只是每一轮中每一个样例的权重会发生变化，权值根据上一次的分类结果进行调整。至于这个权值怎么确定，我们是根据上一次的错误率调整，如果错误率越大则权重越大。最后预测时我们对于每一个分类器的权重是根据错误率进行调整，错误率小的权重大。

常见的集成方法有：

Bagging+决策树=随机森林

AdaBoost+决策树=提升树

Gradient Boosting+决策树=GBDT

AdaBoost

AdaBoost是一种比较常见的集成方法，算法思想如下：

计算样本权重：

假设有n个样本的训练集，设定它们的权重一样，都是1/n：

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) \ldots, (x_n, y_n)\}$$

计算错误率：

计算算法权重：

更新样本权重：

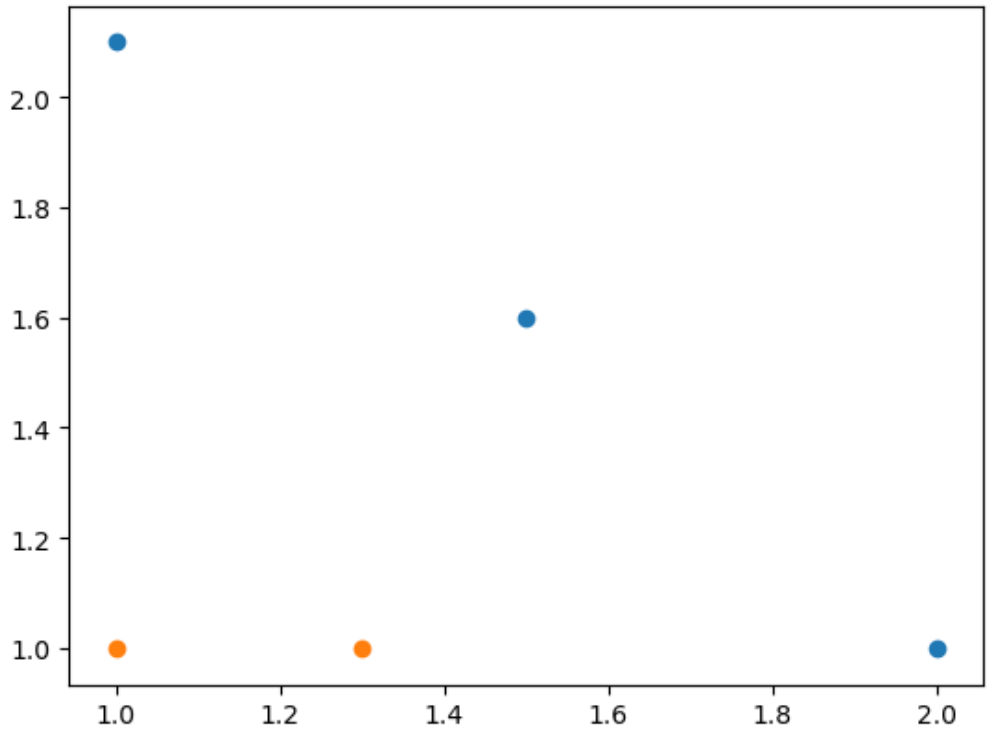
在上一次学习后，需要调整样本的权重，使得第一分类中被错分的样本在下一次学习中得到重点学习：

其中除以的是归一化因子：

也就是说，我们可以化简为：

输出：

我们尝试使用单层决策树对数据集进行分类：




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

def loadData():
    dataMat=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 dataMat,classLabels

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()

def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    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):
    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')
    for i in range(n):
        rangeMin=dataMatrix[:,i].min()
        rangeMax=dataMatrix[:,i].max()
        stepSize=(rangeMax-rangeMin)/numSteps
        for j in  range(-1,int(numSteps)+1):
            for inequal in ['lt','gt']:
                threshVal=(rangeMin+float(j)*stepSize)
                predictedVals=stumpClassify(dataMatrix,i,threshVal,inequal)
                errArr=np.mat(np.ones((m,1)))
                errArr[predictedVals==labelMat]=0
                weightedError=D.T*errArr
                print("split:dim %d, thresh %.2f,thresh ineuqal: %s,the weighted
error is %.3f" % (i,threshVal,inequal,weightedError))
                if weightedError<minError:
                    minError=weightedError
                    bestClasEst=predictedVals.copy()
                    bestStump['dim']=i
                    bestStump['thresh']=threshVal
                    bestStump['ineq']=inequal
    return bestStump,minError,bestClasEst

dataArr,classLabels=loadData()
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)
showDataSet(dataArr,classLabels)
```

> {'dim': 0, 'thresh': 1.3, 'ineq': 'lt'}
> minError:
> [[0.2]]
> bestClasEst:

```
[[-1.]
 [ 1.]
 [-1.]
 [-1.]
 [ 1.]]
```

可以发现，单层决策树是不足以解决问题或者提高精度的，因此我们需要进行ADABOOSTING提升精度，通过迭代几个弱分类器的性能从而提高我们对于输入的分类，因此有：

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

def loadData():
    dataMat=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 dataMat,classLabels

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()

def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    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):
    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')
    for i in range(n):
```

```
        rangeMin=dataMatrix[:,i].min()
        rangeMax=dataMatrix[:,i].max()
        stepSize=(rangeMax-rangeMin)/numSteps
        for j in  range(-1,int(numSteps)+1):
            for inequal in ['lt','gt']:
                threshVal=(rangeMin+float(j)*stepSize)
                predictedVals=stumpClassify(dataMatrix,i,threshVal,inequal)
                errArr=np.mat(np.ones((m,1)))
                errArr[predictedVals==labelMat]=0
                weightedError=D.T*errArr
                print("split:dim %d, thresh %.2f,thresh ineuqal: %s,the weighted
error is %.3f" % (i,threshVal,inequal,weightedError))
                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):
    weakClassArr=[]
    m =np.shape(dataArr)[0]
    D=np.mat(np.ones((m,1))/m)
    aggClassEst=np.mat(np.zeros((m,1)))
    for i in range(numIt):
        bestStump,error,classEst=buildStump(dataArr,classLabels,D)
        #对于每一个样本的权值的赋予
        alpha=float(0.5*np.log((1.0-error)/max(error,1e-16)))
        bestStump['alpha']=alpha
        weakClassArr.append(bestStump)
        expon=np.multiply(-1*alpha*np.mat(classLabels).T,classEst)
        D=np.multiply(D,np.exp(expon))
        D=D/D.sum()
        aggClassEst+=alpha*classEst

aggErrors=np.multiply(np.sign(aggClassEst)!=np.mat(classLabels).T,np.ones((m,1))
)
        errRate=aggErrors.sum()/m
        if errRate==0.0:
            break
    return weakClassArr,aggClassEst

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)):
        classEst=stumpClassify(dataMatrix,classifierArr[i]
['dim'],classifierArr[i]['thresh'],classifierArr[i]['ineq'])

        aggClassEst+=classifierArr[i]['alpha']*classEst
        print(aggClassEst)
    return np.sign(aggClassEst)
```

```
dataArr,classLabels=loadData()
"""
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)
showDataSet(dataArr,classLabels)
"""
weakClassArr,aggClassEst=adaBoostTrainDS(dataArr,classLabels)
print(adaClassify([[0,0],[5,5]],weakClassArr))
```

```
split:dim 1, thresh 2.10,thresh ineuqal: lt,the weighted error is 0.857
split:dim 1, thresh 2.10,thresh ineuqal: gt,the weighted error is 0.143
[[-0.69314718]
 [ 0.69314718]]
[[-1.66610226]
 [ 1.66610226]]
[[-2.56198199]
 [ 2.56198199]]
[[-1.]
 [ 1.]]
```

下面我们使用一个数据集，就是我们之前使用KNN处理的数据集来使用sklearn的adaboost进行处理模拟：

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

def loadData(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
dataArr,classLabels=loadData('horseColicTraining.txt')
testArr,testLabelArr=loadData('horseColicTest.txt')
bdt=AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),algorithm="SAMME",n_e
stimators=10)
bdt.fit(dataArr,classLabels)
predictions=bdt.predict(dataArr)
errArr=np.mat(np.ones((len(dataArr),1)))
```

```
print('训练集错误率：%.3f%%'
%float(errArr[predictions!=classLabels].sum()/len(dataArr)*100))
predictions=bdt.predict(testArr)
errArr=np.mat(np.ones((len(testArr),1)))
print('测试集错误率：%.3f%%'
%float(errArr[predictions!=testLabelArr].sum()/len(testArr)*100))
```

> 训练集错误率：16.054%
> 测试集错误率：17.910%