机器学习——分类器性能评价

在评估分类器性能的时候，我们有一个常见的名词叫做混淆矩阵，也叫误差矩阵，例如我们对于猫和不是猫的分类中：

		预测	**预测**
		猫	不是猫
真实	猫	10	3
真实	不是猫	8	45

上面的列表就是一个混淆矩阵，如果非对角元素都是0的话，那这个分类器的性能是就是完美的。由此对于各种二分类的问题中，对应上面的分类我们就有了一个经典的评估体系：

	预测	**预测**
真实	真正例（TP）	伪反例（FN）
真实	伪正例（FP）	真反例（TN）

对应的我们有以下参数的规定：

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

这个参数越大说明模型的进度越高，也叫作模型的精度。这是对于总体的分类的一个评估，而如果我们需要对阳性预测值进行一个评估，就有了正确率（PPV）：

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

伪发现率（FDR）：

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

错误遗漏率(FOR)：

$$FOR = \frac{FN}{FN + TN}$$

阴性预测值（NPV）：

$$NPV = \frac{TN}{FN + TN}$$

上面这些指标看起来就已经乱了？其实不会，你只要盯紧预测值是同一类的，然后根据他们真实值是哪一个分类进行计算就可以了。接下来就不一样了，接下来就是对于真实值是一样的，进行分类计算，最普遍的就是召回率：

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

召回率的意义就是说明正类样本被正确预测的比例，如果越高就说明预测的求全率越好。同样的有一个假正率，也就是负类样本中被错误预测的正类的比例：

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

基于上面的基础，我们有了一个曲线，叫做ROC曲线，该曲线的横坐标是假正率（FALL），纵坐标就是召回率（RECALL），ROC给出了当阈值发生变化时，假正率与召回率的关系，在最佳关系中，ROC曲线应该尽可能地处于左上角，意味着分类器在假正率很低时获得很高的召回率。例如在过滤垃圾邮件中，我们过滤了所有垃圾邮件，但是又没有将合法邮件归类到垃圾邮件中。

对于ROC曲线的画法，我们首先需要得到所有样本的概率输出，将所有正样本的概率值进行从大到小的排序，然后将这些正样本概率作为阈值，大于该阈值的分类视为正样本，否则视为负样本。

对于画出来的ROC曲线，我们可以根据每一阶梯累加矩形高度得到y轴的和，然后乘以固定x轴的阶梯步进，就可以得到总的面积，就是AUC面积。

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

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

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 plotRoc(predicStrengths,classLabels):
    cur=(1.0,1.0)
    ySum=0.0
```

```
    numPosClas=np.sum(np.array(classLabels)==1.0)
    yStep=1/float(numPosClas)
    xStep=1/float(len(classLabels)-numPosClas)
    sortedIndicties=predicStrengths.argsort()
    fig=plt.figure()
    fig.clf()
    ax=plt.subplot(111)
    for index in sortedIndicties.tolist()[0]:
        if classLabels[index]==1.0:
            delX=0
            delY=yStep
        else:
            delX=xStep
            delY=0
            ySum+=cur[1]
        ax.plot([cur[0],cur[0]-delX],[cur[1],cur[1]-delY])
        cur=(cur[0]-delX,cur[1]-delY)
    plt.title('ROC')
    plt.xlabel('Fall')
    plt.ylabel('Recall')
    ax.axis([0,1,0,1])
    print('Auc面积: ',ySum*xStep)
    plt.show()

dataArr,LabelArr=loadData('horseColicTraining2.txt')
weakClassArr,aggClassEst=adaBoostTrainDS(dataArr,LabelArr,50)
plotRoc(aggClassEst.T,LabelArr)
```